Deckhouse Kubernetes Platform Administrator's guide

Version v1.76

Generated: September 01, 2026

Contents

Contents 2
Deckhouse Kubernetes Platform documentation 71
Installation 71
- Installation methods 71
- Installation options 71
- Installation requirements 71
- Preparing the Configuration 71
  - Installation configuration file 71
  - Post-bootstrap script 76
- Installing 77
  - Pre-installation checks 78
  - Aborting the installation 80
- Air-gapped environment, working via proxy and using external registries 80
  - Installing from an external (third-party) registry 80
  - Nexus configuration notes 82
  - Harbor configuration notes 87
  - Manual loading of DKP images and vulnerability DB into a private registry 88
  - Creating a cluster and running DKP without using release channels 90
  - Using a proxy server 91
  - Automatic proxy variable loading for users in CLI 91
- Additional resources 92

How to configure? 92
- Deckhouse configuration 92
  - Modifying cluster configuration 93
    - Modifying protected parameters 93
      - Modifying protected parameters using dhctl 94
      - Manually changing protected parameters 94
  - Viewing current configuration 95
- Configuring the module 95
  - Enabling and disabling the module 96
- Module bundles 96
  - Things to keep in mind when working with the Minimal module set 97
  - Accessing documentation for the current version 98
- Managing placement of Deckhouse components 98
  - Advanced scheduling 98
  - Module features that depend on its type 98
- Additional resources 98

Working with the image registry 99
- Working with image registries 99
- Working with editions 99

Switching a running DKP cluster to use an external registry 99
- Additional resources 100

Managing the internal container image registry 100
- Modes of operation with the internal registry 100
- Restrictions on working with the internal registry 101
  - Operating conditions restrictions 101
  - Mode switching restrictions 101
- Examples of switching 101
  - Switching to Direct Mode 101
  - Switching to Unmanaged Mode 102
- Migration to the registry module 103
  - For containerd v2 103
  - For Containerd v1 104
- Migration back from the registry module 106
- Check registry mode switch status 109
- Additional resources 110

Restoring registry access when license token is expired or invalid 110

Switching editions 111
- Switching DKP to CE/BE/SE/SE+/EE 111
  - Switching using the registry module 111

Switching without using the registry module 115
Additional resources 117
Authentication 117
Additional resources 118
Integration with external authentication providers 118
General integration workflow 118
OIDC (OpenID Connect) integration 119
Keycloak 119
Blitz Identity Provider 120
Okta 121
SAML 2.0 integration 121
LDAP integration 122
GitHub integration 123
GitLab integration 123
Atlassian Crowd integration 124
Bitbucket Cloud Integration 124
Additional resources 125
Local authentication 125
Creating a static user 125
Local user operations 125
Administrative operations 125
Self-service password reset 126
Adding a user to a group 126
Configuring password policy 126
Configuring two-factor authentication (2FA) 127
Additional resources 127
Accessing the Kubernetes API via load balancer 127
How API access protection works in Kubernetes 128
Access using Basic Authentication (LDAP) 128
Kerberos (SPNEGO) SSO for LDAP 129
Additional resources 130
Resource configuration 130
Example configuration 130
Additional resources 131
Authorization 131
Who gets access and when 131
User authentication 132
Service account authentication 132
Additional resources 132
Current authorization model 132
High-level roles used in the current model 132
Default access list for each high-level role 133
AuthorizationRule example 134
ClusterAuthorizationRule example 134
Extending permissions for high-level roles 135
Additional resources 136
Experimental authorization model 136
Use roles 137
Manage roles 137
Role model subsystems 138
Role model subsystem composition 138
Creating a new subsystem role 139
Label descriptions 139
Aggregation selector descriptions 139
Extending a custom role 140
Extending existing manage-subsystem roles 140
Extending manage-subsystem roles and adding a namespace 141
Extending existing use roles 141
Additional resources 142
Cluster admin access model 142
Kubeconfig files on master nodes 142
RBAC-based admin access 142

- Recommended admin access 142
- Root kubeconfig symlink 143
- Security hardening 143
- Break-glass access 143
- Additional resources 143

Granting permissions to users and service accounts 143
- Granting permissions using AuthorizationRule and ClusterAuthorizationRule (current role model) 143
  - Granting permissions to a user within a single namespace 144
  - Granting permissions to a user in all namespaces 144
- Granting permissions using ClusterRoleBinding and RoleBinding (experimental role model) 144
  - Assigning cluster administrator permissions (experimental role model) 144
  - Assigning networking administrator permissions (experimental role model) 145
  - Granting administrator permissions to a user within a namespace (experimental role model) 146
- Additional resources 147

Checking user permissions 147
- Additional resources 148

Access for CI/CD 148
- ServiceAccount 148
  - Prerequisites 148
  - Create ServiceAccount and a long-lived token 149
  - Grant permissions 149
  - Get API URL and CA certificate 149
  - Create kubeconfig 150
  - Short-lived tokens (TokenRequest) 150
    - Usage patterns in CI 150
- Basic Auth 150
  - Prerequisites 151
  - Enable 151
  - Get API endpoint 151
  - Verification 151
  - Configure kubeconfig 151
  - Use in GitLab CI 151
- Token Exchange 151
  - Prerequisites 152
  - Create DexClient 152
  - Get Dex and API URLs 152
  - Grant RBAC 152
  - Token exchange request 153
  - GitLab CI 153
  - Keycloak 154
  - Diagnostics 154
- Additional resources 155

Integration with IaaS 155

Public cloud providers 155

Connection and authorization in Yandex Cloud 155
- Preparing the environment 155
- Creating a service account 156
- Assigning IAM roles 156
- Generating an authorization key 156
- Checking and increasing quotas 156
- Reserving a public IP 156
- Additional resources 157

Layouts and configuration in Yandex Cloud 157
- Layouts 157
  - Standard 157
  - WithoutNAT 158
  - WithNATInstance 160
- Defining the YandexClusterConfiguration 162
  - Configuration example 163
- Network parameters and security 163

Internal node addressing 163
Assigning external IP addresses and outgoing traffic 164
DNS and DHCP settings for internal networks 164
Using pre-existing subnets 164
Additional external networks 165
Configuring security groups in Yandex Cloud 165
Configuring access via bastion host 165
Additional resources 166
Storage and load balancing in Yandex Cloud 166
Storage (CSI and StorageClass) 166
Excluding unnecessary StorageClasses 166
Setting the default StorageClass 167
Resizing a PVC 167
Load balancing 169
External LoadBalancer 169
Internal LoadBalancer 169
Service annotations 170
Using a separate target group for a NodeGroup 170
Target group health checks 171
Applying changes 171
Changing networkType does not recreate CloudEphemeral nodes 171
Integrating manually created VMs 172
Additional resources 172
Integration with Yandex Cloud services 172
Integration with Yandex Lockbox 172
Integration with Yandex Managed Service for Prometheus 174
Additional resources 175
Connection and authorization in VK Cloud 175
Layouts and configuration in VK Cloud 175
Layouts 175
Standard 175
StandardWithNoRouter 177
Simple 179
SimpleWithInternalNetwork 181
Configuration 183
Configuration examples 184
Example 1 184
Example 2 184
List of required services 184
LoadBalancer configuration 185
IngressNginxController example 185
Node security policies and configuration 185
Assigning additional security groups to static and master nodes 185
Assigning additional security groups to ephemeral nodes 185
Uploading an image to VK Cloud 186
Verifying security group support by the cloud provider 187
Configuring online volume resize 187
Using rootDiskSize 188
Disks in VK Cloud 188
rootDiskSize parameter 188
Recommendations for master nodes and bastion with network-attached disks 188
Recommendations for ephemeral nodes with local disks 188
Checking the disk size in a flavor 188
Overriding the cloud provider’s default disk type 189
Connection and authorization in Selectel Cloud 189
Layouts and configuration in Selectel Cloud 189
Layouts 189
Standard 189
StandardWithNoRouter 191
Simple 193
SimpleWithInternalNetwork 195
Configuration 197

Configuration examples 198
Example 1 198
Example 2 198
List of required services 198
LoadBalancer configuration 199
IngressNginxController example 199
Node security policies and configuration 199
Assigning additional security groups to static and master nodes 199
Assigning additional security groups to ephemeral nodes 199
Uploading an image to Selectel 200
Verifying security group support by the cloud provider 201
Configuring online volume resize 201
Using rootDiskSize 202
Disks in Selectel 202
rootDiskSize parameter 202
Recommendations for master nodes and bastion with network-attached disks 202
Recommendations for ephemeral nodes with local disks 202
Checking the disk size in a flavor 202
Overriding the cloud provider’s default disk type 203
Offline disk resizing 203
Connection and authorization in OVH Cloud 203
Layouts and configuration in OVH Cloud 203
Layouts 203
Standard 203
StandardWithNoRouter 205
Simple 207
SimpleWithInternalNetwork 209
Configuration 211
Configuration examples 212
Example 1 212
Example 2 212
List of required services 212
LoadBalancer configuration 213
IngressNginxController example 213
Node security policies and configuration 213
Assigning additional security groups to static and master nodes 213
Assigning additional security groups to ephemeral nodes 213
Uploading an image to OVHcloud 214
Verifying security group support by the cloud provider 215
Configuring online volume resize 215
Using rootDiskSize 216
Disks in OVHcloud 216
rootDiskSize parameter 216
Recommendations for master nodes and bastion with network-attached disks 216
Recommendations for ephemeral nodes with local disks 216
Checking the disk size in a flavor 216
Overriding the cloud provider’s default disk type 217
Offline disk resizing 217
Connection and authorization in OpenStack 217
Layouts and configuration in OpenStack 217
Layouts 217
Standard 217
StandardWithNoRouter 219
Simple 221
SimpleWithInternalNetwork 223
Configuration 225
Configuration examples 226
Example 1 226
Example 2 226
List of required services 226
LoadBalancer configuration 227
IngressNginxController example 227

Node security policies and configuration 227
Assigning additional security groups to static and master nodes 227
Assigning additional security groups to ephemeral nodes 227
Uploading an image to OpenStack 228
Verifying security group support by the cloud provider 229
Configuring online volume resize 229
Using rootDiskSize 230
Disks in OpenStack 230
rootDiskSize parameter 230
Recommendations for master nodes and bastion with network-attached disks 230
Recommendations for ephemeral nodes with local disks 230
Checking the disk size in a flavor 230
Overriding the cloud provider’s default disk type 231
Offline disk resizing 231
Connection and authorization in Amazon Web Services 231
Requirements 231
Accessing the AWS API 231
Configuring IAM via the AWS web console 233
Configuring IAM via AWS CLI 234
Configuring IAM via Terraform 235
Using a custom IAM role for nodes (iamNodeRole) 235
Additional resources 236
Layouts and configuration in Amazon Web Services 236
WithoutNAT 236
WithNAT 237
Defining AWSClusterConfiguration 239
Internal addressing and subnets 240
Security groups 240
Setting up VPC peering 240
Configuring access via bastion host 240
Creating a cluster in a new VPC with an existing bastion host 241
Creating a cluster in a new VPC and deploying a bastion host 241
Using an existing VPC (existingVPCID) 241
Additional resources 241
Storage and load balancing in Amazon Web Services 241
Storage (CSI and StorageClass) 241
Expanding volume size 242
Load balancing 242
Configuring LoadBalancer when Ingress nodes are not in all zones 242
Connecting CloudStatic nodes 243
Additional resources 243
Integration with Amazon Web Services 243
Working with Amazon Elastic Container Registry (ECR) 243
Additional resources 244
Connection and authorization in Microsoft Azure 244
Requirements 244
Accessing the Azure API 244
Additional resources 244
Layouts and configuration in Microsoft Azure 244
Standard 245
Storage and load balancing in Microsoft Azure 245
Storage 245
Load balancing 246
Additional resources 246
Integration with Microsoft Azure services 246
Support for Service Endpoints 246
Additional resources 247
Connection and authorization in Google Cloud Platform 247
Creating a service account 247
Setup via Google Cloud console 247
Setup via Google Cloud CLI 247
Using the service account 248

- Additional resources 248
- Layouts and configuration in Google Cloud Platform 248
  - Layouts 248
    - Standard 248
    - WithoutNAT 250
  - Configuration 252
    - Enabling nested virtualization 253
    - Adding additional disks 253
    - Configuring node security policies 254
    - Setting additional network tags for static and master nodes 254
    - Setting additional network tags for ephemeral nodes 254
    - Adding CloudStatic nodes to the cluster 254
  - Additional resources 254
- Private cloud providers 254
- Connection and authorization in VK Cloud 254
- Layouts and configuration in VK Cloud 256
  - Layouts 256
    - Standard 256
    - StandardWithNoRouter 257
    - Simple 259
    - SimpleWithInternalNetwork 261
  - Configuration 263
    - Configuration examples 264
      - Example 1 264
      - Example 2 264
    - List of required services 264
    - LoadBalancer configuration 265
      - IngressNginxController example 265
    - Node security policies and configuration 265
      - Assigning additional security groups to static and master nodes 265
      - Assigning additional security groups to ephemeral nodes 265
    - Uploading an image to VK Private Cloud 266
    - Verifying security group support by the cloud provider 267
    - Configuring online volume resize 267
    - Using rootDiskSize 268
      - Disks in VK Private Cloud 268
      - rootDiskSize parameter 268
        - Recommendations for master nodes and bastion with network-attached disks 268
        - Recommendations for ephemeral nodes with local disks 268
      - Checking the disk size in a flavor 268
    - Overriding the cloud provider’s default disk type 269
- Connection and authorization in OpenStack 269
- Layouts and configuration in OpenStack 269
  - Layouts 269
    - Standard 269
    - StandardWithNoRouter 271
    - Simple 273
    - SimpleWithInternalNetwork 275
  - Configuration 277
    - Configuration examples 278
      - Example 1 278
      - Example 2 278
    - List of required services 278
    - LoadBalancer configuration 279
      - IngressNginxController example 279
    - Node security policies and configuration 279
      - Assigning additional security groups to static and master nodes 279
      - Assigning additional security groups to ephemeral nodes 279
    - Uploading an image to OpenStack 280
    - Verifying security group support by the cloud provider 281
    - Configuring online volume resize 281
    - Using rootDiskSize 282

Disks in OpenStack 282
rootDiskSize parameter 282
Recommendations for master nodes and bastion with network-attached disks 282
Recommendations for ephemeral nodes with local disks 282
Checking the disk size in a flavor 282
Overriding the cloud provider’s default disk type 283
Offline disk resizing 283
Connection and authorization in Huawei Cloud 283
Requirements 283
Accessing the Huawei Cloud API 283
Creating a user group 283
Creating an IAM user 283
JSON policies 284
Layouts and configuration in Huawei Cloud 286
Standard 286
VpcPeering 287
Storage and load balancing in Huawei Cloud 288
Storage 288
Load balancing 288
Additional resources 288
Integration with Huawei Cloud services 288
Working with InstanceClass 288
Additional resources 289
Virtualization systems 289
Integration with Deckhouse Virtualization Platform cloud 289
Additional resources 289
Connection and authorization in Deckhouse Virtualization Platform 289
Creating a user 289
Assigning a role 290
Generating a kubeconfig 290
Layouts and configuration in Deckhouse Virtualization Platform 290
Standard 290
Configuration 291
Additional resources 293
Connection and authorization in VMware Cloud Director 294
Resource preparation 294
User permissions 294
Adding a network 297
DHCP setup 300
Adding a vApp 302
Adding a network to a vApp 303
Incoming traffic 305
Configuring DNAT/SNAT rules on the Edge Gateway 305
Let’s Encrypt certificate issuance 306
Firewall setup 306
Virtual machine template 308
Preparing the template from an OVA file 309
Configuring the template in VCD 313
Storage usage 318
Using the LoadBalancer 319
Additional resources 319
Layouts and configuration in VMware Cloud Director 319
Layouts 319
Standard 319
WithNAT 320
Configuration 322
Example configuration of VCDInstanceClass 323
Storage 323
CSI 323
Additional resources 323
Connection and authorization in VMware vSphere 323
Requirements 323

VM image requirements 324
Preparing the virtual machine image 324
vSphere configuration 325
Configuration in vSphere Client 325
Creating tags and tag categories in vSphere Client 325
Configuring Datastore in vSphere Client 328
Creating and assigning a role in vSphere Client 329
Configuration with govc 330
Installing govc 330
Creating tags and tag categories with govc 330
Datastore configuration with govc 331
Creating and assigning a role with govc 331
Additional resources 332
Layouts and configuration in VMware vSphere 332
Standard 332
List of required privileges 334
Additional resources 337
Storage and load balancing in VMware vSphere 337
Storage 337
Resizing a volume (PVCs) 337
Load balancing 337
CSI 338
Important information on PVC size expansion 338
Datastore configuration 338
Additional resources 338
Integration with VMware vSphere services 338
vSphere resource management 338
Removing CloudPermanent nodes in vSphere 338
Reducing the number of nodes 339
Completely removing a node group 339
Additional resources 340
Connection and authorization in zVirt 340
Requirements 340
Preparing a cloud image 340
Preparing a virtual machine template 341
Layouts and configuration in zVirt 343
Standard 344
Additional resources 345
Storage and load balancing in zVirt 345
Storage 345
Requirements 345
Configuration 345
Load balancing 345
Additional resources 345
Integration with zVirt services 345
Key features 345
Additional resources 346
Hybrid clusters 346
NodeGroup types 346
Ways to add nodes 346
General network requirements 346
Additional resources 347
Hybrid cluster with Yandex Cloud 347
Prerequisites 347
Adding automatically created nodes 347
Adding manually created nodes through a bootstrap script 351
Additional resources 353
Hybrid cluster with VCD 353
Prerequisites for VCD 353
Adding automatically created nodes 353
Adding manually created nodes through a bootstrap script 354
Additional resources 356

- Hybrid cluster with vSphere 356
  - Prerequisites for vSphere 356
  - Adding automatically created nodes 356
  - Adding manually created nodes through a bootstrap script 358
  - Additional resources 360
- Hybrid cluster with DVP 360
  - Prerequisites 360
  - Adding automatically created nodes 361
  - Adding manually created nodes using a bootstrap script 363
  - Additional resources 364
- High reliability and availability 364
  - Additional resources 364
- Recommendations for configuring cluster nodes and preventing overload 364
  - Master nodes 365
  - Frontend nodes 365
  - Monitoring nodes 365
  - System nodes 365
  - Preventing Kubernetes API overload (FlowSchema) 365
  - Delaying node reboot or shutdown while critical pods are running 365
    - Enabling a node shutdown delay 365
    - Searching pods that prevent shutdown and performing the node troubleshooting 366
  - Additional resources 366
- Managing HA mode 366
  - Enabling HA mode globally 366
    - Using ModuleConfig/global custom resource 366
    - Using Deckhouse web UI 367
  - Configuring HA mode with two master nodes and an arbiter node 367
    - Configuring in a cloud cluster 367
    - Configuring in a static cluster 368
  - Enabling HA mode for individual components 368
  - Additional resources 369
- Chaos engineering 369
  - Additional resources 370
- Platform scaling and node management 370
- General management and configuration of the control plane 370
  - Main features 370
  - Enabling, disabling, and configuring the module 370
    - Enabling / disabling 370
    - Configuration 371
    - Checking DKP status and queues 371
  - Certificate management 372
    - PKI management 372
    - Additional SANs and auto-update 372
    - Kubelet certificate rotation 372
    - Manual renewal of control plane certificates 372
  - Speeding up pod restarts after losing connection to a node 373
  - Forcibly disabling IPv6 on cluster nodes 373
  - Additional resources 373
- Control plane recovery and debugging 374
  - Recovery from failures 374
  - Restoring etcd cluster functionality 374
  - Restoring a master node when kubelet fails to load control plane components 374
    - containerd 374
  - etcd restore 374
    - Viewing etcd cluster members 374
      - Option 1 374
      - Option 2 374
    - Restoring the etcd cluster in case of complete unavailability 375
    - Recovering etcd after panic: unexpected removal of unknown remote peer error 375
    - Actions to take when etcd database exceeds quota-backend-bytes limit 375
  - etcd defragmentation 376
    - How to defragment an etcd node in a single-master cluster 376

How to defragment etcd in a cluster with multiple master nodes 377
High availability 378
Additional resources 378
Scaling and changing master nodes 378
Scaling and single/multi-master transition 378
Control plane operation modes 378
Automatic scaling of master nodes 379
Common scaling scenarios 379
Removing the master role from a node without deleting the node itself 379
Changing the OS image of master nodes in a multi-master cluster 380
In a cloud cluster 380
In a static cluster 381
Changing the OS image in a single-master cluster 383
In a cloud cluster 383
In a static cluster 383
Adding master nodes to a static or hybrid cluster 383
Adding master nodes in a cloud cluster 384
Reducing the number of master nodes in a cloud cluster 385
Accessing the DKP controller in a multi-master cluster 386
Additional resources 386
Node management basics in Deckhouse 386
Automatic deployment and updates 387
Basic node and OS configuration 387
Disruptive updates 387
Example of a system NodeGroup 388
Example NodeGroupConfiguration descriptions 388
Installing the cert-manager plugin for kubectl on master nodes 388
Werf configuration for ignoring the Ready status of a node group 388
Settings for Static and CloudStatic NodeGroups 388
Running DKP on an arbitrary node 389
Additional resources 389
Adding and managing cloud nodes 389
Adding CloudEphemeral nodes in a cloud cluster 389
Configuration for CloudEphemeral NodeGroups 390
Modifying the cloud provider configuration in a cluster 390
NodeGroup autoscaling 390
Example autoscaling scenario 391
Allocating nodes for specific workloads 391
Accelerating node provisioning in the cloud during horizontal application scaling 392
Example NodeGroup definition 392
Cloud nodes 392
Configuring CloudEphemeral nodes using NodeGroupConfiguration 392
Example NodeGroupConfiguration definitions 393
Setting a sysctl parameter 393
Adding a root certificate to the host 393
Adding a certificate to the OS and containerd 394
Kernel update on nodes 395
Debian-based distros 395
CentOS-based distros 396
Adding CloudPermanent nodes to a cloud cluster 397
Adding a CloudStatic node to a cluster 398
Manually 398
Using the Cluster API Provider Static 398
Using Cluster API Provider Static for multiple node groups 400
Adding master nodes in a cloud cluster 401
Using NodeGroup with priority 401
NodeGroup states and their interpretation 402
Additional resources 403
Adding and managing bare-metal nodes 403
Adding nodes to a bare-metal cluster 403
Manual method 403
Automatic method 404

Modifying a static cluster configuration 405
Moving a static node between NodeGroups 405
Manual cleanup of a static node 406
NodeGroup example 406
Example NodeGroup definition for static nodes 406
Changing the CRI for a NodeGroup 406
Changing the NodeGroup of a static node 407
Changing the IP address in a StaticInstance 407
Deleting a StaticInstance 407
Additional resources 407
Custom node configuration 407
Monitoring script execution 409
Script re-execution mechanism 409
Script writing specifics 410
How to use containerd with Nvidia GPU support 410
Ubuntu 411
CentOS 412
MIG in a single-node cluster 412
Verifying successful installation 413
How to deploy a custom containerd configuration file 414
Adding a configuration for an additional registry 415
Old Method 416
Adding authorization to an additional registry (deprecated method) 416
Configuring a certificate for an additional registry (deprecated method) 417
Adding TLS skip verify (deprecated method) 418
Configuring a mirror for access to public registries (deprecated method) 419
New Method 420
Adding authorization to an additional registry (actual method) 421
Configuring a certificate for an additional registry (actual method) 421
Adding TLS skip verify (actual method) 422
Configuring a mirror for access to public registries (actual method) 423
How to automatically put custom labels on the node 424
Additional resources 424
Migrating container runtime to containerd v2 424
Requirements 424
How to enable containerd v2 425
Additional resources 425
Application scaling and resource management 426
Scaling by metrics 426
Scaling based on metrics 426
Configuration 426
Configuring metrics using observability module resources 427
Working with unstable metrics 427
Retrieving metric values 428
Configuring autoscaling in the Deckhouse Web Interface 428
Additional resources 429
Horizontal pod autoscaling 429
How Horizontal Scaling (HPA) works 429
Available metric types for HPA 429
HPA recommendations 429
HPA limitations 429
How to enable or disable HPA 429
HPA configuration 429
HPA configuration examples 430
Scaling based on CPU and memory 430
Scaling based on object metrics 430
Scaling based on external metrics 431
Additional resources 432
Vertical pod autoscaling 432
How Vertical Scaling (VPA) works 432
How to enable or disable VPA 432
VPA configuration 432

Module configuration example 433
Additional resources 433
Pod restart on configuration change 433
Supported annotations 433
Usage examples 433
Tracking all changes in all attached resources 433
Tracking changes in specific resources 434
Tracking changes in resources from the list 435
Enabling or disabling pod restart 435
Configuration 435
Additional resources 436
Scheduler 436
Pod scheduling 436
General overview of the scheduler 436
Pod scheduling phases 436
How to modify or extend scheduler logic 437
Additional pod placement mechanisms 437
Scheduler profiles 438
Pod redistribution 438
Considering pod Priority Classes 438
How DKP considers pod priority 438
Which pods are not evicted 439
How to enable or disable pod eviction 439
Descheduling strategies 439
Additional resources 441
Priority classes 442
Pod priorities (Priority Classes) 442
Available Priority Classes 442
Network 443
Configuring and managing incoming traffic 443
Managing outgoing traffic 443
Internal network configuration 443
Configuring inter-cluster communication 443
Network policies 443
Additional resources 443
Internal network configuration 443
Internal network operation modes 443
Mode configuration 444
Services 444
Supported service types 444
Configuring the service network with kube-proxy 444
Additional resources 445
Encrypting traffic between pods 445
mTLS configuration example 445
Additional resources 447
Configuring cluster network stack visualization 448
Additional resources 448
Switching CNI from Flannel or Simple bridge to Cilium 448
Incoming traffic balancing 449
Network-level load balancing (NLB) 449
Application-level load balancing (ALB) 449
Additional resources 449
Balancing with MetalLB 450
Layer 2 mode 450
How it works 450
Advantages over classic MetalLB 450
Behavior comparison 450
Example of using MetalLB in L2 LoadBalancer mode 450
BGP mode 452
How it works 452
Configuration 452
Establishing BGP sessions 452

Assigning IP addresses to Services 452
Announcing IP addresses 452
Traffic distribution 452
Benefits of using BGP 452
Drawbacks of using BGP 452
Example of using MetalLB in BGP LoadBalancer mode 452
Additional resources 453
Balancing in clusters on cloud platforms 453
Example of creating an Ingress controller for the OpenStack provider 453
Example of creating a ClusterIP-type Service 453
Example for VK Cloud 454
ALB with Ingress NGINX Controller 454
Traffic termination options 454
HTTPS termination 454
Monitoring and statistics 454
Statistics 455
Basic principles of collecting statistics 455
Metrics structure and representation 455
Load balancing configuration examples 455
Example for AWS (Network Load Balancer) 455
Example for GCP / Yandex Cloud / Azure 456
Example for OpenStack 456
Example for VK Cloud 456
Example for bare metal 456
Example for bare metal with external load balancer 457
Example for bare metal (MetalLB in BGP LoadBalancer mode) 457
Example for bare metal (MetalLB in L2 LoadBalancer mode) 457
Example of segregating access between public and administrative zones 458
Specifics of using a single Ingress controller 459
Using separate Ingress controllers 459
Module support and security 461
Additional resources 461
ALB with Kubernetes Gateway API 462
Validating Gateway API configuration 462
Steps to take before enabling and configuring ALB in a cluster 462
Using with other modules and third-party solutions 462
Using third-party Gateway API solutions 462
Publishing an application 463
Creating a Gateway object 463
Creating ListenerSet objects to manage the handling of incoming requests 464
Creating routes and configuring routing 465
Publishing service domains 466
Algorithm for selecting the default DKP gateway when using multiple ClusterALBInstances 466
Changing the default DKP gateway 467
Changing the inlet while keeping the current Gateway 467
Opening an additional TCP port on the cluster-wide gateway 467
Viewing Envoy Proxy Configuration 468
Additional resources 468
ALB with Istio 468
Ingress to publish applications 468
Istio Ingress Gateway 468
Ingress NGINX 470
Additional resources 471
Ensuring high availability and fault tolerance (keepalived) 471
Module usage examples 471
Multiple public IP addresses 471
Switching keepalived manually 473
Additional resources 473
Egress Gateway configuration 473
Egress Gateway usage modes 473
Basic mode 473
Virtual IP mode 473

Configuration principle 473
Node maintenance 473
Comparison with CiliumEgressGatewayPolicy 474
Configuration examples 474
EgressGateway in PrimaryIPFromEgressGatewayNodeInterface mode (basic mode) 474
EgressGateway in VirtualIPAddress mode 474
EgressGatewayPolicy 475
Additional resources 475
Inter-cluster cooperation (cluster alliance) 475
Service mesh usage specifics in DKP 475
Differences between federation and multicluster 476
Models for combining multiple clusters (Istio example) 476
Problems solved with multicluster and federation 476
Additional resources 476
Multicluster 476
Multicluster of Istio funds 476
Requirements for clusters 476
General principles 477
Enabling the multicluster 477
Managing the multicluster 477
Example of configuring a multicluster of two clusters 477
Additional resources 478
Federation 478
Federation with Istio (Service Mesh) 478
Requirements for clusters 478
General principles of federation 478
Enabling the federation — created services 478
Managing the federation 478
Example of configuring a federation of two clusters 479
Additional resources 479
Network policy configuration 479
Configuring network policies via standard Kubernetes means 479
Example network policy configuration 479
Configuring cluster-wide network policies using CiliumClusterwideNetworkPolicy 480
Additional resources 481
Software-defined networking (SDN) 481
Additional resources 481
Configuring SDN in a cluster 481
Preparing the infrastructure for enabling the sdn module 481
Enabling the sdn module 482
Actions after enabling the sdn module 482
Marking interfaces for organizing additional software-defined networks 482
Combining multiple physical interfaces into a channel aggregation interface (bond interface) 482
Configuring and connecting additional virtual networks for use in application pods 484
Creating a publicly accessible network (cluster) 484
Creating a network based on tagged traffic 484
Creating a network based on direct interface access 485
Creating a project network (user network) 486
Checking the connection of an additional network to interfaces on nodes 486
IPAM: IP address pools for additional networks 487
Principles and features of IPAM in DKP 487
Protection against conflicts when reusing IP addresses 488
Resources and settings for configuring IPAM 488
Example of assigning IP addresses to the cluster network 488
Configuring and connecting underlay networks for hardware device passthrough 489
Key features 489
Operation modes 489
Automatic interface grouping 489
Procedure for configuring and connecting physical interfaces to application subnets 489
Prerequisites for DPDK applications 490
Configuring hugepages 490

Configuring Topology Manager 490
Preliminary steps before creating an UnderlayNetwork 490
Checking and configuring network interfaces (NodeNetworkInterface) 491
Creating UnderlayNetwork in Dedicated mode 491
Creating UnderlayNetwork in Shared mode 492
Preparing namespaces for UnderlayNetwork usage 493
Configuring and connecting system networks (service networks) 494
Prerequisites for creating and using system networks 494
Creating a system network 494
VLAN type 494
Access type 495
SRIOVVirtualFunction type 495
Creating a pool of IP addresses for configuring the IPAM system network 495
Checking system network status 496
Additional resources 496
Managing static routes and ip rules on cluster nodes 496
Configuration examples 496
Creating a route in main routing table 496
Creating a route in additional routing table 497
Creating an ip rule 497
Additional resources 498
Managing DNS in a Kubernetes cluster 498
Configuration example 499
Additional resources 499
Caching DNS requests on cluster nodes 499
Example custom DNS configuration in a Pod 499
Additional resources 499
BpfLB operation modes for external traffic processing 500
Additional resources 501
Synchronizing time on nodes 501
Enabling built-in time synchronization 501
Using custom NTP daemons 502
Additional resources 502
Access via OpenVPN 502
Configuration 502
External access to OpenVPN 503
Example for bare-metal clusters 503
Example for AWS and Google Cloud 503
Example for a public IP address on an external Load Balancer 503
Authentication 503
Available cluster resources after connecting to VPN 504
Auditing user connections 504
Why automatic DNS configuration doesn’t work on macOS and Linux with the OpenVPN client 504
How to revoke, rotate or delete a user certificate 504
How to rotate a server certificate 505
How to rotate a root certificate (CA) 505
Additional resources 505
Storage 506
Supported storage types 506
Software-defined storage 506
External storage 506
Key features 506
Storage configuration 506
Creating a StorageClass 506
Setting a default StorageClass 506
Additional resources 507
Configuring volume snapshot creation 507
Supported CSI drivers 507
Creating snapshots 507
Restoring from a snapshot 507
Additional resources 508
Local path provisioner storage 508

How it works 508
Limitations 508
Example LocalPathProvisioner resources 508
ReclaimPolicy: Retain (default) 508
ReclaimPolicy: Delete 509
Configuring Prometheus to use local storage 509
Additional resources 509
Node configuration for software-defined storage 509
Additional resources 510
Usage 510
Working with BlockDevice resources 510
Creating a BlockDevice resource 510
Criteria for device selection by the controller 510
Updating a BlockDevice resource 510
Deleting a BlockDevice resource 510
Working with LVMVolumeGroup resources 511
Creating an LVMVolumeGroup resource 511
Updating an LVMVolumeGroup resource 513
Deleting an LVMVolumeGroup resource 513
Removing a BlockDevice from an LVMVolumeGroup 513
Protecting against data leaks between volumes 514
Thick volume 514
Using Discard mode 514
Thin volume 514
Additional resources 514
Configuration scenarios 515
Cloning virtual machines 515
Methods and scenarios for configuring the disk subsystem of nodes 515
Storage with identical disks 516
Full mirror 516
Example of configuring SDS modules (identical disks, “Full mirror”) 517
Configuring the sds-local-volume module (identical disks, “Full mirror”) 517
Configuring the sds-replicated-volume module (identical disks, “Full mirror”) 517
Partial mirror 518
Example of configuring SDS modules (identical disks, “Partial mirror”) 519
Configuring the sds-local-volume module (identical disks, “Partial mirror”) 519
Configuring the sds-replicated-volume module (identical disks, “Partial mirror”) 520
Combined storage 521
Configuring additional disks (Full mirror) 522
Example of configuring SDS modules (combined storage, “Full mirror”) 522
Configuring the sds-local-volume module (combined storage, “Full mirror”) 522
Configuring the sds-replicated-volume module (combined storage, “Full mirror”) 523
Configuring additional disks (Partial mirror) 524
Example of configuring SDS modules (Combined storage, “Partial mirror”) 525
Configuring the sds-local-volume module (combined storage, “Partial mirror”) 525
Configuring the sds-replicated-volume module (combined storage, “Partial mirror”) 526
Additional resources 528
FAQ 529
Reasons why BlockDevice resources aren’t created in the cluster 529
Reasons why LVMVolumeGroup resources aren’t created in the cluster 529
Reasons why after deleting the LVMVolumeGroup, the resource and Volume Group remain 529
Reasons for failed attempts to create a Volume Group using the LVMVolumeGroup resource 529
Behavior of the LVMVolumeGroup resource when one of the devices in the Volume Group is disabled 529
Transferring management of an existing Volume Group on the node to the controller 529
Disabling tracking of the LVM Volume Group by the controller 530
Reasons for automatic setting of the tag storage.deckhouse.io/enabled=true on the Volume Group 530
Using the LVMVolumeGroupSet resource to create LVMVolumeGroup 530
Labels added by the controller to BlockDevice resources 530
Additional resources 531
Setting up local storage based on LVM 531

Configuring local storage 531
Quick start 531
Enabling modules 531
Preparing nodes for storage creation 532
Configuring storage on nodes 532
Selecting a volume cleanup method after PV deletion 535
Thick volumes 536
Thin volumes 536
System requirements and recommendations 536
Additional resources 536
Managing local storage based on LVM 536
Selecting specific nodes for module usage 536
Verifying PVC creation on the selected node 537
Removing a node from the module management 537
Verifying dependent LVMVolumeGroup resources on the node 538
Remaining sds-local-volume-csi-node Pod after removing labels 539
Absence of component service Pods on the desired node 539
Data migration between PVCs 540
Creating volume snapshots 541
Additional resources 541
Setting up replicated storage based on DRBD 541
Enabling the module 542
Discovery of LVM components 542
Connecting DRBD 542
Node pre-configuration 543
Creating LVM volume groups 543
Creating replicated thick pools 545
Creating replicated thin pools 545
Creating StorageClass objects 546
Additional resources 547
Managing DRBD-based replicated storage 547
System requirements 547
Recommendations 547
Retrieving space-usage information 547
Assigning the default StorageClass 548
Adding an existing LVMVolumeGroup 548
Changing DRBD volume limits and cluster ports 548
Correctly rebooting a node with DRBD resources 548
Moving resources to free up space in a Storage Pool 549
Automatic management of replicas and monitoring of LINSTOR state 549
Evicting DRBD resources from a node 550
Preparing and running the script 550
Example of removing a node from LINSTOR and Kubernetes 550
Example of removing resources from a node 550
Parameters of the evict.sh script 551
Troubleshooting 551
Start error of linstor-node while loading the DRBD module 551
FailedMount error when starting a Pod 552
Pod stuck in ContainerCreating 552
Input/output error 552
After the ReplicatedStoragePool resource is deleted, the corresponding Storage Pool remains in the backend 552
Restrictions on changing ReplicatedStorageClass spec 552
Deleting a child StorageClass when deleting a ReplicatedStorageClass 552
Errors when creating a Storage Pool or StorageClass 552
Error “You’re not allowed to change state of linstor cluster manually” 552
Restoring the database from a backup 553
Backup restoration procedure 553
Missing sds-replicated-volume service Pods on a selected node 555
Additional support 556
Migration from the linstor module to sds-replicated-volume 556
Migration procedure 556

Migrating to ReplicatedStorageClass 557
Migrating to ReplicatedStoragePool 558
Migration from the sds-drbd module to sds-replicated-volume 558
Migration procedure 558
Reasons to avoid RAID with sds-replicated-volume 559
Recommendations for using local disks 559
Manual trigger the certificate renewal process 559
Additional resources 560
Distributed Ceph storage 560
Migration from ceph-csi module 560
Connecting to a Ceph cluster 561
Additional information 562
How to get a list of RBD volumes separated by nodes 562
Which versions of Ceph clusters are supported 563
Which volume access modes are supported 563
Examples of permissions (caps) for Ceph users 563
RBD 563
CephFS 563
CephFS + RBD 563
Additional resources 564
HPE data storage 564
System requirements 564
Configuration 564
Enabling the module 564
Creating a StorageClass 564
Module health verification 565
Additional resources 565
Huawei data storage 565
System requirements 566
Configuration 566
Enabling the module 566
Creating a StorageClass 566
Module health verification 567
Additional resources 567
NetApp data storage 567
System Requirements 568
Configuring cluster integration with the NetApp storage system 568
Additional resources 569
NFS storage 569
System Requirements 569
Minimum Requirements 569
Configuration Recommendations 569
Limitations 569
General Limitations 569
Volume Snapshot Limitations 570
RPC-with-TLS Mode Limitations 570
Functional Limitations 570
System Requirements 570
Configuration 570
Enabling the Module 570
Creating StorageClass 571
Volume Cleanup Configuration 571
Parameter RandomFillSinglePass 571
Parameter RandomFillThreePass 571
Parameter Discard 572
Changing NFS Server Parameters for Created PersistentVolumes 572
Creating Volume Snapshots 572
Issues with Deleting PVs with RPC-with-TLS Support 573
Adding Multiple CAs to the tlsParameters.ca Parameter 573
Additional resources 573
S3 object storage 573
System requirements 573

- Configuration 574
  - Enabling the module 574
  - Creating a StorageClass 574
- Module health check 575
- Limitations 575
- Known bugs 575
- Changing connection parameters for existing PVs 575
- Displaying the mounted directory size 575
- Handling exceeded bucket or user quotas 576
- Obtaining information about used space 576
- Using multiple S3 storages in a single module 576
- Using a single bucket for multiple Pods 576
- Troubleshooting 576
  - Issues creating a PVC 576
  - Issues starting containers 576
- Additional resources 576

SCSI-based data storage 577
- Supported features 577
- Limitations 577
- System requirements 577
- Quick start 577
  - Enabling the csi-scsi-generic module 577
  - Creating an SCSITarget 578
    - iSCSI 578
    - Fibre Channel 579
  - Creating a StorageClass 580
  - Csi-scsi-generic module health check 581
- Additional resources 581

TATLIN.UNIFIED (Yadro) unified storage 581
- Enabling the module 581
- Connecting to the TATLIN.UNIFIED storage system 581
- Creating a StorageClass 582
- Ensuring the module works 582
- Additional resources 582

Security 582
- Additional resources 583

Security events 583
- Dependencies and requirements 583
- Data sources for security events 583
- Enabling security events 584
- Configuring event sources 584
- Configuring event delivery 585
  - Configuring a destination 585
  - Configuring event delivery rules 586
- Default module settings 586
- Supported security events 587
- Additional resources 587

Kubernetes API event audit 587
- Built-in audit policies 587
- Configuring a custom audit policy 588
- How to generate the contents of the audit-policy.yaml file 589
  - Examples 590
    - Log all requests from all authenticated users 590
    - Logging all requests not from service accounts 590
    - Logging all operations with standard resources 591
- Working with the audit log file 591
- Redirecting the audit log file to stdout 591
- Additional resources 592

Runtime audit 592
- Data sources for runtime audit 592
- Minimum requirements 592
- Enabling runtime audit 592

Manually configuring the Kubernetes API audit webhook 593
Working with audit rules 593
Adding a custom rule 594
Applying a third-party rule 594
Log collection and alerts 595
Configuring log and event collection 595
Configuring critical event alerts 596
Viewing metrics 596
Debugging and simulating events 597
Enabling debug logging 597
Simulating events 597
Falco 597
Falcosidekick 597
Additional resources 598
Security policies 598
Applying Pod Security Standards 598
Default policy 598
Assigning a policy 598
Enforcement modes 599
Extending a policy 599
Operational policies 601
Security policies 601
Partial policy enforcement 604
Gatekeeper custom resources 606
Custom Gatekeeper policy examples 606
Preventing deletion of a node without a specified label 606
Preventing exec and attach operations to specific Pods 607
Built-in policy for heritage: deckhouse label on Pods 607
Custom policy example 608
Image signature verification 609
Using alternative security policy management tools 611
Additional resources 611
Container image vulnerability scanning 611
Vulnerability scanning 611
Scanning in namespaces 611
Start conditions and scanning process 611
Forcing a rescan 612
Certificate management 612
Monitoring 612
Access roles 612
Working with certificate issuers 613
Adding a ClusterIssuer with DNS-01 validation via webhook 613
Adding a ClusterIssuer that uses a custom certificate authority (CA) 614
Adding an Issuer and ClusterIssuer using HashiCorp Vault to request certificates 615
Additional resources 616
Integration with software and security solutions 616
Sending logs to KUMA 616
Example ClusterLogDestination and ClusterLoggingConfig configurations 616
KCS configuration features 620
Antivirus scanning exclusions for nodes 620
KESL configuration recommendations 620
Additional resources 621
Automatic distribution of secrets across namespaces 621
How it works 621
Incompatibility with multitenancy mode 621
Configuring automatic secret distribution 621
Distributing secrets to selected namespaces 621
Monitoring in Deckhouse Kubernetes Platform 621
Hardware resource monitoring 622
Kubernetes monitoring 622
Ingress monitoring 622
Disabling collection of detailed statistics from Ingress resources 622

- Control plane monitoring 623
- Cluster monitoring 623
- Extended monitoring mode 623
- Alerts 623
  - Sending alerts to external systems 623
- Cluster SLA monitoring 624
- Additional resources 624

Configuring network interaction and cluster node monitoring 624
- Network interaction monitoring 624
  - Adding additional IP addresses for monitoring 624
- Cluster node monitoring 625
- Additional resources 625

Writing Prometheus data to longterm storage 625
- Example of minimal PrometheusRemoteWrite 625
- Example of advanced PrometheusRemoteWrite 625
- Additional resources 626

External access configuration 626
- Connecting Prometheus to Third-Party Grafana 626
- Connecting Third-Party Applications to Prometheus 628
- Metrics Collection via Gateway (Pushgateway) 629
  - Removing Metrics from Gateway (Pushgateway) 630
- Additional resources 631

Configuring a system for collecting and storing metrics 631
- What does Prometheus do? 631
- How does Prometheus work? 631
- Additional resources 631

Cluster SLA monitoring 631
- Module configuration 632
- Interface 632
- Status metrics export 633
- Authentication 633
- Behavior of upmeter pods 634
- Additional resources 634

Configuring integrations 634
- Redirecting alerts to Zabbix 634
  - Installation 634
  - Troubleshooting 637
- Sending alerts to Telegram 638
- Example of sending alerts to Slack with filter 638
- Example of sending alerts to Opsgenie 639
- Example of sending alerts via email 640
- Additional resources 641

Logging 641

Log collection and delivery 641
- Configuring log collection and delivery 642
- Integration with external systems 643
  - Elasticsearch 643
    - Using index templates 643
    - Support for Elasticsearch < 6.X 644
  - Splunk 644
  - Logstash 645
  - Graylog 645
- Message formats 645
  - Syslog 646
  - CEF 646
- Log transformation 646
  - Converting records to structured objects 647
  - Label replacement 647
  - Label removal 648
- Log filtering 649
  - Collecting logs from a specific container 649
  - Collecting logs without a specific string 649

Kubelet audit events 650
DKP system logs 650
Log buffering 651
Example of enabling in-memory buffering 651
Example of enabling disk buffering 651
Example of defining buffer overflow behavior 651
Debugging and advanced features 652
Enabling debug logs for the log-shipper agent 652
Additional information about log transmission channels 652
Topology overview 652
Monitoring channel load 653
Getting raw and intermediate logs 653
Debugging VRL rules 653
Adding support for new source or sink 653
Special cases 654
Collecting logs from production namespaces using the labelSelector option 654
Label for excluding pods and namespaces 654
Additional resources 654
Short-term log storage 654
Integration with Grafana Cloud 655
Migration from Grafana Promtail 655
Additional resources 655
Multitenancy 655
Limitations 656
Additional resources 656
Project management 656
Project creation 656
Automatic project creation for a namespace 657
Creating a custom project template 658
Using labels to manage resources 658
Skipping creation of the heritage: multitenancy-manager label 658
Excluding resources from management by multitenancy-manager 658
Implementing validation of object changes with a custom label 659
How validation works in multitenancy-manager 659
Creating your own validation 659
Granting cluster-scoped resources to projects 661
Additional resources 662
Backup and restore 662
Manual cluster restore 662
Restoring a cluster with a single control plane node 662
Restoring a multi-master cluster 664
Restoring individual objects 664
Restoring Kubernetes objects from an etcd backup 664
Example of steps to restore objects from an etcd backup 665
Restoring cluster objects from exported YAML files 667
Restoring objects after changing the master node IP address 667
Automated object extraction when changing IP address 668
Manual object restore after changing the IP address 669
Creating backups with Deckhouse CLI 670
Backing up etcd 671
Automatic etcd backup 671
Cluster configuration backup 671
Exporting logs from Loki 673
Additional resources 674
Automatic assignment of namespace labels and annotations 674
How it works 674
Configuring automatic label and annotation assignment 674
Example configuration 674
Additional resources 675
Application delivery with Argo CD 675
Additional resources 675
Running Argo CD 677

Preparing to run Argo CD 677
Deploying an Argo CD instance 678
Deploying multiple Argo CD instances 679
Advanced settings 679
Enabling high availability mode 679
Granting access to cluster resources 680
Using a custom cluster domain 682
Additional resources 682
Configuring authentication and authorization 682
Local authentication 682
Creating additional local users 682
Creating a user token 683
Disabling local authentication 683
SSO authentication 683
Using a self-signed certificate 685
Creating a user token 685
Additional resources 686
Configuring role-based access control 686
Built-in roles 686
Default policy for authenticated users 687
Anonymous access 688
Role-based access control structure 688
Role binding 688
Access policy 688
Resources and supported actions 688
Application-bound policies 689
The applications resource 689
Detailed permissions for update and delete 689
The action action 690
The override action 690
Other resources 690
The applicationsets resource 690
The logs resource 690
The exec resource 690
The extensions resource 690
The deny effect 691
Policy evaluation order and matching modes 691
Matching in glob mode 691
Using SSO users and groups 691
Local users 692
Testing RBAC policies 692
Additional resources 692
Marketplace 692
Administrator tasks 693
Key resources 693
Additional resources 693
Package repositories 693
Authentication and Scanning Interval Management 693
Authentication 693
Scan interval 694
Checking repository status 694
Viewing discovered package versions 695
Additional resources 695
Scanning 695
Viewing scan operations 695
Inspecting a scan operation 696
Triggering a manual scan 696
fullScan flag 696
Additional resources 697
Platform update 697
Configuring updates 697
Checking the current release channel 697

- Switching release channels 697
- Update modes 698
  - Patch versions only within the current minor (AutoPatch) 698
  - Automatic updates for all versions (Auto) 698
  - Manual mode (Manual) 698
  - Checking the current update mode 698
  - How automatic updates are applied 699
    - Release pinning 699
  - Manual update approval 700
- Update windows 700
  - Applying updates when update windows are configured 700
  - Configuring update windows 700
    - Configuration examples 701
- Additional resources 701

Configuring notifications on new releases 701
- Alerts in the monitoring system 701
- Configuring notifications 702
  - Supported notifications parameters 703
- Notification format 704
- Additional resources 705

Updating Kubernetes and versioning 705
- Updating and version management 705
  - Changing the Kubernetes version 705
- Monitoring Kubernetes update progress 705
  - ConfigMap content examples 706
    - Cluster up to date (3 master nodes, 3 worker nodes) 706
    - Start of update (e.g., Kubernetes version downgrade) 707
    - Control plane update in progress 708
    - Intermediate step of a multi-hop upgrade (e.g. 1.33 → 1.35) 709
    - Cluster up to date (2 master nodes, 1 arbitr node and 3 worker nodes) 710
    - Failure of one or more control plane components 711
- Additional resources 712

Uninstallation 712
- Deleting a cluster deployed with a cloud provider 712
- Deleting a hybrid cluster 713
- Deleting a static cluster 714
- Additional resources 714

Architecture 715
- DKP architecture 715
- Modules 715
- Additional resources 715

DKP architecture (C4 model) 715
- Legend 716
- Context diagram 716
- Subsystem diagram 717
- Module diagram 718
- Additional resources 719

Deckhouse Kubernetes Platform module development 719
- Requirements 720
- Before you start 720
- Got a question? 720

Module structure 720
- charts 721
- crds 721
- docs 722
- hooks 723
- images 724
- openapi 724
  - conversions 724
    - Conversion tests 725
  - config-values.yaml 725
    - Validations with x-deckhouse-validations 725

Examples of rules 725
Scalar and array value validation 726
Validation of additionalProperties (map) 727
values.yaml 727
templates 727
.helmignore 728
Chart.yaml 728
module.yaml 728
Configuring module accessibility in DKP editions 730
Module accessibility logic 731
Configuration examples 731
Additional resources 731
Build and publish 731
Types of module artifacts 732
Building module artifacts and publishing them to the container registry 732
Additional resources 733
How to run and verify a module in the DKP cluster 733
Run the module in the DKP cluster 733
Module source 733
Switching the module to a different module source 737
Module update policy 737
Enabling the module 738
Troubleshooting 739
Integrating Deckhouse Module Tools for Module Validation 739
For GitHub Projects 739
For GitLab Projects 740
Steps to connect 740
Additional resources 740
Dependencies of the Deckhouse Kubernetes Platform modules 740
Deckhouse Kubernetes Platform version dependency 740
Kubernetes version dependency 741
Dependency on the version of other modules 742
Required dependencies 742
Conditional module requirements 743
Restrictions on enabling and disabling prometheus when there is an conditional dependency on test 743
Restrictions on updating prometheus when there is an conditional dependency on test 743
Restrictions on enabling test, whose version is specified in prometheus dependencies 744
Restrictions on updating test, whose version is specified in prometheus dependencies 744
Versioning modules 744
Release channels 744
Procedure for publishing a new version 744
Module lifecycle 744
Definition of module stability 745
API versioning 746
Release of the new API version 746
Automatic API Conversion 746
Recommendations for Releasing New Versions of CRD 746
Additional resources 746
An example of creating a module with adaptation of an existing chart 747
Preparing module source code and building 747
Publishing the module to the release channel 748
Enabling a module in a cluster 749
Migration of ModuleUpdatePolicy to version v1alpha2 750
Additional resources 751
Module development and debugging 751
Module availability and enabling by default 752
How ModulePullOverride works 753
An example 753
Module auto-update logic 755
Skipping intermediate releases (from-to) 755
Examples 757

- Module artifacts in the container registry 758
  - Print the list of modules in the module source 759
  - Print the list of module images 759
  - Print the list of files in the module image 759
  - Print the list of images of the module's application containers 759
  - Configuring extra images 759
  - Print the list of releases 760
  - Print the version in use on the alpha release channel 760
- Additional resources 760

Disaster resilience 760

Geo-distribution 760
- Using multiple availability zones (Multi-AZ) 761
  - Availability zone features 761
  - Distributing nodes across zones 761
- Using multiple regions (Multi-Region) 761
  - Region features 761
  - Distributing nodes across regions 761
- Balancing incoming traffic 761
- Storage organization 761

Geo-reserving 762
- Balancing incoming traffic in a multi-cluster 762
  - Balancing traffic via active-active method 762
  - Balancing traffic via active-standby method 762
- Storage organization in a multi-cluster 763

Updating 763
- Releases 763
- Release channels 763
  - Switching to a more stable channel 764
  - Switching to a less stable channel 764
- Control plane updates 764
  - Patch version updates 764
  - Automatic minor version updates 764
  - Manual minor version updates 764
  - Process of control plane component updates 764
- Retrieving the changelog 765
  - Changelog contents 765
  - Minor versions and zero patch releases 765
- Checking dependencies before update 765
- Additional resources 765

Deckhouse subsystem 765

Web interface modules 766
- Console module 766
  - Module architecture 766
  - Module components 766
  - Module interactions 767
- Documentation module 767
  - Module architecture 767
  - Module components 768
  - Module interactions 768
- Deckhouse-tools module 769
  - Module architecture 769
  - Module components 769
  - Module interactions 769
- Additional resources 770

Deckhouse module 770
- Module architecture 770
- Module components 770
- Module interactions 771
- Additional resources 772

Queuing mechanism 772
- Addon-operator queues 772
- Marketplace queues 773

Additional resources 774
Registry module 774
Module architecture 775
Module components 778
Module interactions 779
Additional resources 779
Direct mode architecture 779
Additional resources 780
Marketplace 780
Sections 780
Concepts 780
Package types 780
Resource model 781
ApplicationPackageVersion content 781
Scan-to-deploy lifecycle 781
Application constraints 781
Functional constraints 781
Naming constraints 782
Additional resources 782
Application development 782
Prerequisites 782
Bootstrapping a new Application 782
Project structure 782
package.yaml 783
Local build 784
Linting 784
CI/CD setup 784
Environment variables 784
Triggering a release 784
OCI artifact layout in the registry 784
Bundle contents 784
Version metadata image contents 784
Nelm annotations 785
Deployment stages 785
1. Render 785
2. Plan 785
3. Apply 785
Common use cases 785
1. Job that must run before the main application 785
2. Preserve a resource across uninstall or chart removal 786
3. Resource shared between releases 786
4. Wait for a resource created by an operator 786
5. Non-critical component that must not fail the deploy 787
6. Resource rendered only on first install 787
7. Slow-starting resource with a large image 787
Ordering and dependencies 787
Weights 787
Direct dependencies 787
External dependencies 788
Delete dependencies 788
Lifecycle annotations 788
helm.sh/resource-policy 788
werf.io/ownership 788
werf.io/deploy-on 788
werf.io/delete-policy 789
werf.io/delete-propagation 789
Tracking annotations 789
Log annotations 789
Template functions 790
werf_secret_file 790
dump_debug, printf_debug, include_debug, tpl_debug 790
Known pitfalls 790

Hooks 790
ApplicationHookInput 790
Example: basic hook 790
Settings validation hook 791
Namespace isolation guarantee 791
Delivery subsystem 792
Operator-argo module 792
Module architecture 792
Module components 792
Module interactions 795
Additional resources 796
Kubernetes & Scheduling subsystem 796
Cluster control plane 796
Kubelet 796
Static pod manifests 796
Kubelet interactions 796
Additional resources 798
Cluster control plane management 798
Control-plane-manager module 798
Module architecture 798
Module components 799
Module interactions 799
Cluster control plane monitoring 800
Components of the monitoring-kubernetes-control-plane module 800
Interactions of control-plane-proxy component 800
Metrics collection from kube-apiserver 800
Additional resources 800
Vertical Pod Autoscaler 800
VPA operating modes 800
VPA limitations 801
Module architecture 801
Module components 802
Module interactions 803
Additional resources 803
Adaptive-resource-management module 803
Module architecture 804
Module components 804
Module interactions 804
Additional resources 804
Descheduler module 804
Module architecture 805
Module components 805
Module interactions 805
Additional resources 805
Pod-reloader module 805
Module architecture 805
Module components 806
Module interactions 806
Additional resources 806
Cluster & Infrastructure subsystem 806
Node management 807
Node types and addition mechanics 807
Automatic deployment, configuration, and update of Kubernetes nodes 807
Kubernetes node deployment 807
Maintaining node state 808
Working with nodes in supported clouds 808
Cloud node scaling 808
Working with Static nodes 808
Node grouping and group management 809
What is the Instance resource 809
When node reboot is required 809
NodeGroup parameter effects on node updates and restarts 809

Additional resources 810
Bashible 810
Bashible scripts and the bashible service 810
Bashible-api-server 810
Node-manager module 810
Module architecture 811
Additional resources 811
CloudEphemeral node management 811
Module architecture 811
Module components 811
Module interactions 812
Architecture features specific to CloudEphemeral nodes 813
Additional resources 813
CloudPermanent node management 813
Module architecture 813
Module components 814
Module interactions 814
Architecture features specific to CloudPermanent nodes 815
Additional resources 815
CloudStatic node management 815
Module architecture 815
Module components 816
Module interactions 817
Architecture features specific to CloudStatic nodes 817
Additional resources 817
Static node management 817
Module architecture 817
Module components 818
Module interactions 819
Architecture features specific to Static nodes 819
Additional resources 819
Hybrid node group and cluster management 819
Additional resources 820
Cloud-provider-aws module 820
Module architecture 820
Module components 821
Module interactions 822
Additional resources 822
Cloud-provider-azure module 823
Module architecture 823
Module components 823
Module interactions 824
Additional resources 824
Cloud-provider-dvp module 824
Module architecture 825
Module components 825
Module interactions 826
Additional resources 827
Cloud-provider-dynamix module 827
Module architecture 827
Module components 828
Module interactions 829
Additional resources 830
Cloud-provider-gcp module 830
Module architecture 830
Module components 830
Module interactions 831
Additional resources 831
Cloud-provider-huaweicloud module 832
Module architecture 832
Module components 833
Module interactions 834

Additional resources 835
Cloud-provider-openstack module 835
Module architecture 835
Module components 836
Module interactions 837
Additional resources 837
Cloud-provider-vcd module 837
Module architecture 837
Module components 838
Module interactions 839
Additional resources 840
Cloud-provider-vsphere module 840
Module architecture 840
Module components 841
Module interactions 842
Additional resources 842
Cloud-provider-yandex module 842
Module architecture 842
Module components 844
Module interactions 844
Additional resources 845
Cloud-provider-zvirt module 845
Module architecture 845
Module components 846
Module interactions 847
Additional resources 848
Chrony module 848
Module architecture 848
Module components 849
Module interactions 849
Additional resources 850
Gpu module 850
DRA mode 850
Module architecture 850
Module components 850
Module interactions 851
Device Plugin mode 851
Module architecture 852
Module components 852
Module interactions 853
Additional resources 853
Terraform-manager module 853
Module architecture 853
Module components 854
Module interactions 855
Additional resources 855
IAM subsystem 855
Authentication 855
Connecting to Kubernetes API using a generated kubeconfig 855
Protection against credential brute-forcing in Dex 855
Authentication via DexAuthenticator 855
DKP-specific Dex extensions 856
High availability mode 856
Additional resources 856
Multitenancy 856
Internal logic 856
Project creation 856
Project isolation 856
Module architecture 857
Module components 857
Module interactions 857
Additional resources 857

User-authn module 857
Module architecture 858
Module components 860
Module interactions 860
Additional resources 861
Security subsystem 861
Integrity control 861
Integrity control of user workloads at startup 861
Integrity control of running containers 862
Integrity protection of DKP modules 862
Additional resources 862
Runtime audit architecture 862
Audit rules 863
Built-in rules 863
Custom rules 863
Additional resources 863
Security events architecture 863
Solution architecture 863
Processing pipeline 864
Collection layer (log-shipper) 864
Processing and extraction 864
Transformation and enrichment 865
Filtering and delivery 865
Security event model 865
Minimal sufficient lifecycle 866
Supported security events 866
Additional resources 866
Admission-policy-engine module 866
Module architecture 867
Module components 868
Module interactions 868
Additional resources 868
Cert-manager module 868
Module architecture 868
Module components 869
Module interactions 870
Additional resources 870
Module operator-trivy 870
Module architecture 871
Module components 871
Module interactions 873
Additional resources 873
Runtime-audit-engine module 873
Module architecture 874
Module components 874
Module interactions 874
Additional resources 875
Secrets-store-integration module 875
Module architecture 875
Module components 875
Module interactions 876
Additional resources 877
Network subsystem 877
Cni-cilium module 877
Module architecture 877
Module components 878
Module interactions 880
Additional resources 880
Cilium-hubble module 880
Module architecture 880
Module components 881
Module interactions 881

Additional resources 882
Ingress-nginx module 882
Module architecture 882
Module components 882
Module interactions 883
Receiving traffic from external networks 884
Architecture of Ingress controller with HostWithFailover inlet type 884
Components of failover Ingress controller 885
Failover Ingress controller interactions 885
Additional resources 885
Metallb module 885
Module architecture 886
Module components 887
Module interactions 887
Additional resources 888
DNS modules 889
kube-dns module 889
Module architecture 889
Module components 889
Module interactions 890
node-local-dns module 890
When using cni-cilium 890
Module architecture 890
Module components 891
Module interactions 891
When using cni-flannel or cni-simple-bridge 892
Module architecture 892
Module components 892
Module interactions 892
Additional resources 893
Cluster-level Istio architecture 893
Estimating overhead 894
Control plane 894
Data plane 894
Additional resources 894
Application service architecture with Istio enabled 894
Architecture features 894
User request lifecycle 895
Application with Istio disabled 895
Application with Istio enabled 895
Additional resources 895
Caching DNS server in a cluster 895
Caching DNS server operation principles 895
CoreDNS configuration aspects 895
Grafana dashboard 895
Additional resources 896
Storage subsystem 896
Local-path-provisioner module 896
Module architecture 896
Module components 897
Module interactions 897
Additional resources 897
Snapshot-controller module 897
Module architecture 898
Module components 898
Module interactions 899
Additional resources 899
Sds-local-volume module 899
Module architecture 899
Module components 900
Module interactions 901
Additional resources 901

Sds-node-configurator module 901
Module architecture 901
Module components 903
Module interactions 903
Additional resources 903
Csi-ceph module 903
Module architecture 903
Module components 904
Module interactions 905
Additional resources 905
Csi-hpe module 905
Module architecture 905
Module components 906
Module interactions 907
Additional resources 907
Csi-huawei module 907
Module architecture 907
Module components 908
Module interactions 908
Additional resources 909
Csi-netapp module 909
Module architecture 909
Module components 910
Module interactions 910
Additional resources 910
Csi-nfs module 910
Module architecture 910
Module components 911
Module interactions 912
Additional resources 912
Csi-s3 module 912
Module architecture 912
Module components 913
Module interactions 913
Additional resources 914
Csi-scsi-generic module 914
Module architecture 914
Module components 915
Module interactions 916
Additional resources 916
Csi-vsphere module 916
Module architecture 916
Module components 917
Module interactions 917
Additional resources 917
Csi-yadro-tatlin-unified module 917
Module architecture 918
Module components 918
Module interactions 919
Additional resources 919
Standard CSI driver 919
Driver architecture 919
Driver components 920
Driver interactions 921
Additional resources 921
Ceph RBD CSI driver 921
Driver architecture 922
Driver components 922
Driver interactions 923
Additional resources 923
Huawei CSI driver 923
Driver architecture 923

Driver components 924
Driver interactions 925
Additional resources 925
NFS CSI driver 925
Driver architecture 925
Driver components 926
Driver interactions 927
Additional resources 927
S3 CSI driver 927
Driver architecture 927
Driver components 928
Driver interactions 929
SCSI Generic CSI driver 929
Driver architecture 929
Driver components 930
Driver interactions 931
VMware vSphere CSI driver 931
Driver architecture 931
Driver components 932
Driver interactions 933
Additional resources 933
Observability subsystem 934
Architecture of monitoring in the Deckhouse Kubernetes Platform 934
Composition and interaction scheme of monitoring components 934
Components installed by DKP 934
Prometheus 935
What’s in the Prometheus pod? 935
How is Prometheus configured? 935
Where does Prometheus get the list of targets? 936
How are Service Monitors processed? 937
How are custom resources with rules processed? 938
Architecture of DKP component availability assessment (upmeter) 938
Additional resources 939
Prometheus module 939
Module architecture 939
Module components 939
Module interactions 940
Fault tolerance and High Availability (HA) monitoring mode 941
Additional resources 941
Observability module 941
Module architecture 941
Module components 941
Module interactions 942
Additional resources 942
Upmeter module 943
Module architecture 943
Module components 943
Module interactions 944
Additional resources 944
Logging modules 944
Log-shipper module 944
Module architecture 944
Module components 945
Module interactions 945
Loki module 945
Module architecture 946
Module components 946
Module interactions 946
Additional resources 946
Managed Services subsystem 946
Additional resources 947
Managed-memcached module 947

- Module architecture 947
- Module components 948
- Module interactions 948
- Additional resources 948

Managed-postgres module 948
- Module architecture 948
- Module components 949
- Module interactions 950
- Additional resources 951

Virtualization subsystem 951
- Additional resources 951

The module core 951
- Module core architecture 952
- Module core components 952
- Module core interactions 954
- Additional resources 954

Deckhouse Virtualization Container Registry (DVCR) 954
- DVCR architecture 954
- DVCR components 955
- DVCR interactions 955
- Additional resources 956

Containerized Data Importer (CDI) 956
- CDI architecture 956
- CDI components 957
- CDI interactions 958
- Additional resources 958

Auxiliary components 958
- Security events audit 958
  - Architecture 958
  - Components 959
  - Interactions 959
- Virtualization-dra and other components 959
  - Architecture 959
  - Components 960
  - Interactions 961
- Additional resources 961

Global configuration 961
- Conversions 962
- Parameters 962
- Additional resources 965

Custom Resources 965
- Application 965
- ApplicationPackage 967
- ApplicationPackageVersion 968
- ClusterConfiguration 973
- CNIMigration 976
- CNINodeMigration 977
- DeckhouseRelease 978
- InitConfiguration 979
- Module 980
- ModuleConfig 982
- ModuleDocumentation 983
- ModulePullOverride 984
- ModuleRelease 985
- ModuleSettingsDefinition 987
- ModuleSource 987
- ModuleUpdatePolicy 989
- PackageRepository 993
- PackageRepositoryOperation 995
- SSHConfig 998
- SSHHost 999
- StaticClusterConfiguration 999

Release channels 1000
Additional resources 1000
Supported Kubernetes and OS versions 1001
Linux 1001
Kubernetes 1001
Container registry 1001
Deckhouse Kubernetes Platform editions comparison 1001
Container registry structure 1005
Commercial editions container registry 1006
Network interaction of the platform components 1006
Requirements for network latency between master nodes 1006
Potential issues 1007
Checking current latencies 1007
Network latency at the other nodes 1007
Traffic between master nodes 1007
Traffic from master nodes to regular nodes 1008
Traffic from nodes to master nodes 1008
Traffic between nodes 1008
Static virtualization ports on cluster nodes (4100-4134/TCP reserved) 1012
External traffic to master nodes 1013
External traffic to frontend nodes 1013
External traffic for all nodes 1013
Additional resources 1013
The list of alerts 1015
Alert severity 1015
Module admission-policy-engine 1015
Module cert-manager 1016
Module chrony 1018
Module cloud-provider-huaweicloud 1018
Module cloud-provider-yandex 1018
Module cni-cilium 1020
Module control-plane-manager 1021
Module documentation 1024
Module extended-monitoring 1025
Module ingress-nginx 1036
Module istio 1039
Module kube-dns 1042
Module log-shipper 1042
Module loki 1043
Module metallb 1044
Module monitoring-applications 1045
Module monitoring-custom 1045
Module monitoring-deckhouse 1046
Module monitoring-kubernetes 1054
Module monitoring-kubernetes-control-plane 1060
Module monitoring-ping 1062
Module node-local-dns 1062
Module node-manager 1064
Module okmeter 1072
Module openvpn 1072
Module operator-prometheus 1073
Module prometheus 1074
Module secret-copier 1079
Module terraform-manager 1080
Module upmeter 1082
Module user-authn 1086
Sysctl parameters managed by the platform 1087
Additional resources 1090
Information about used software 1090
Used directors 1095
Deckhouse CLI overview and installation 1096
Main command groups 1096

How do I install the Deckhouse CLI? 1097
trdl-based installation 1097
Installing the executable 1097
Deckhouse CLI command reference 1101
d8 backup 1101
d8 backup cluster-config 1101
d8 backup etcd 1101
d8 backup loki 1102
d8 cli 1102
d8 cli check 1102
d8 cli update 1103
d8 cli use 1103
d8 cli versions 1104
d8 completion 1104
d8 completion bash 1104
d8 completion fish 1104
d8 completion powershell 1104
d8 completion zsh 1105
d8 cr 1105
d8 cr catalog 1105
d8 cr config 1106
d8 cr digest 1106
d8 cr export 1106
d8 cr fs 1107
d8 cr fs cat 1107
d8 cr fs extract 1108
d8 cr fs ls 1108
d8 cr fs tree 1109
d8 cr login 1109
d8 cr ls 1109
d8 cr manifest 1110
d8 cr pull 1110
d8 cr push 1111
d8 data 1111
d8 data create 1111
d8 data delete 1112
d8 data download 1112
d8 data export 1112
d8 data export create 1112
d8 data export delete 1112
d8 data export download 1113
d8 data export list 1113
d8 data import 1113
d8 data import create 1114
d8 data import delete 1114
d8 data import upload 1114
d8 data list 1114
d8 delivery-kit 1115
d8 delivery-kit build 1115
d8 delivery-kit bundle 1120
d8 delivery-kit bundle apply 1120
d8 delivery-kit bundle copy 1126
d8 delivery-kit bundle plan 1128
d8 delivery-kit bundle publish 1133
d8 delivery-kit bundle render 1141
d8 delivery-kit chart 1146
d8 delivery-kit chart ts 1146
d8 delivery-kit chart ts build 1146
d8 delivery-kit chart ts init 1147
d8 delivery-kit ci-env 1149
d8 delivery-kit cleanup 1150
d8 delivery-kit completion 1155

d8 delivery-kit compose 1155
d8 delivery-kit compose config 1155
d8 delivery-kit compose down 1160
d8 delivery-kit compose run 1165
d8 delivery-kit compose up 1171
d8 delivery-kit config 1176
d8 delivery-kit config graph 1176
d8 delivery-kit config list 1177
d8 delivery-kit config render 1179
d8 delivery-kit converge 1180
d8 delivery-kit cr 1190
d8 delivery-kit cr login 1190
d8 delivery-kit cr logout 1191
d8 delivery-kit dismiss 1192
d8 delivery-kit export 1198
d8 delivery-kit helm 1202
d8 delivery-kit helm create 1203
d8 delivery-kit helm dependency 1204
d8 delivery-kit helm dependency build 1205
d8 delivery-kit helm dependency list 1207
d8 delivery-kit helm dependency update 1208
d8 delivery-kit helm env 1209
d8 delivery-kit helm get 1210
d8 delivery-kit helm get all 1211
d8 delivery-kit helm get hooks 1212
d8 delivery-kit helm get manifest 1213
d8 delivery-kit helm get metadata 1214
d8 delivery-kit helm get notes 1215
d8 delivery-kit helm get values 1216
d8 delivery-kit helm get-autogenerated-values 1217
d8 delivery-kit helm get-namespace 1221
d8 delivery-kit helm get-release 1223
d8 delivery-kit helm history 1225
d8 delivery-kit helm install 1226
d8 delivery-kit helm lint 1229
d8 delivery-kit helm list 1231
d8 delivery-kit helm package 1233
d8 delivery-kit helm plugin 1234
d8 delivery-kit helm plugin install 1235
d8 delivery-kit helm plugin list 1236
d8 delivery-kit helm plugin uninstall 1237
d8 delivery-kit helm plugin update 1238
d8 delivery-kit helm pull 1239
d8 delivery-kit helm registry 1241
d8 delivery-kit helm registry login 1242
d8 delivery-kit helm registry logout 1243
d8 delivery-kit helm repo 1244
d8 delivery-kit helm repo add 1245
d8 delivery-kit helm repo index 1246
d8 delivery-kit helm repo list 1247
d8 delivery-kit helm repo remove 1248
d8 delivery-kit helm repo update 1249
d8 delivery-kit helm rollback 1250
d8 delivery-kit helm search 1252
d8 delivery-kit helm search hub 1253
d8 delivery-kit helm search repo 1254
d8 delivery-kit helm secret 1255
d8 delivery-kit helm secret decrypt 1256
d8 delivery-kit helm secret encrypt 1257
d8 delivery-kit helm secret file 1259
d8 delivery-kit helm secret file decrypt 1260
d8 delivery-kit helm secret file edit 1261

d8 delivery-kit helm secret file encrypt 1262
d8 delivery-kit helm secret generate-secret-key 1264
d8 delivery-kit helm secret rotate-secret-key 1265
d8 delivery-kit helm secret values 1266
d8 delivery-kit helm secret values decrypt 1267
d8 delivery-kit helm secret values edit 1269
d8 delivery-kit helm secret values encrypt 1270
d8 delivery-kit helm show 1271
d8 delivery-kit helm show all 1272
d8 delivery-kit helm show chart 1274
d8 delivery-kit helm show crds 1275
d8 delivery-kit helm show readme 1277
d8 delivery-kit helm show values 1278
d8 delivery-kit helm status 1280
d8 delivery-kit helm template 1281
d8 delivery-kit helm test 1285
d8 delivery-kit helm uninstall 1286
d8 delivery-kit helm upgrade 1287
d8 delivery-kit helm verify 1291
d8 delivery-kit helm version 1292
d8 delivery-kit host 1293
d8 delivery-kit host cleanup 1293
d8 delivery-kit host purge 1295
d8 delivery-kit includes 1296
d8 delivery-kit includes get-file 1296
d8 delivery-kit includes ls-files 1298
d8 delivery-kit includes update 1300
d8 delivery-kit kube-run 1301
d8 delivery-kit lint 1306
d8 delivery-kit managed-images 1314
d8 delivery-kit managed-images add 1314
d8 delivery-kit managed-images ls 1317
d8 delivery-kit managed-images rm 1320
d8 delivery-kit plan 1324
d8 delivery-kit purge 1333
d8 delivery-kit render 1337
d8 delivery-kit rollback 1346
d8 delivery-kit run 1350
d8 delivery-kit sbom 1354
d8 delivery-kit sbom get 1354
d8 delivery-kit sbom merge 1360
d8 delivery-kit sbom validate 1361
d8 delivery-kit slugify 1362
d8 delivery-kit stages 1363
d8 delivery-kit stages copy 1363
d8 delivery-kit synchronization 1367
d8 delivery-kit verify 1369
d8 delivery-kit version 1370
d8 help 1370
d8 iam 1370
d8 iam access 1370
d8 iam access grant 1370
d8 iam access revoke 1371
d8 iam get 1372
d8 iam get group 1372
d8 iam get rule 1372
d8 iam get user 1372
d8 iam group 1373
d8 iam group add-member 1373
d8 iam group create 1373
d8 iam group delete 1374
d8 iam group remove-member 1374

d8 iam list 1374
d8 iam list groups 1374
d8 iam list rules 1375
d8 iam list users 1375
d8 iam user 1375
d8 iam user create 1376
d8 iam user delete 1376
d8 iam user lock 1377
d8 iam user reset-password 1377
d8 iam user reset2fa 1378
d8 iam user unlock 1378
d8 k 1378
d8 k alpha 1379
d8 k annotate 1381
d8 k api-resources 1383
d8 k api-versions 1385
d8 k apply 1386
d8 k apply edit-last-applied 1389
d8 k apply set-last-applied 1391
d8 k apply view-last-applied 1393
d8 k attach 1394
d8 k auth 1396
d8 k auth can-i 1397
d8 k auth reconcile 1399
d8 k auth whoami 1400
d8 k autoscale 1402
d8 k certificate 1404
d8 k certificate approve 1406
d8 k certificate deny 1407
d8 k cluster-info 1409
d8 k cluster-info dump 1411
d8 k completion 1412
d8 k config 1414
d8 k config current-context 1415
d8 k config delete-cluster 1416
d8 k config delete-context 1418
d8 k config delete-user 1419
d8 k config get-clusters 1420
d8 k config get-contexts 1422
d8 k config get-users 1423
d8 k config rename-context 1424
d8 k config set 1426
d8 k config set-cluster 1427
d8 k config set-context 1428
d8 k config set-credentials 1430
d8 k config unset 1432
d8 k config use-context 1433
d8 k config view 1434
d8 k cordon 1436
d8 k cp 1437
d8 k create 1439
d8 k create clusterrole 1441
d8 k create clusterrolebinding 1443
d8 k create configmap 1445
d8 k create cronjob 1447
d8 k create deployment 1449
d8 k create ingress 1451
d8 k create job 1453
d8 k create namespace 1455
d8 k create poddisruptionbudget 1457
d8 k create priorityclass 1459
d8 k create quota 1461

d8 k create role 1463
d8 k create rolebinding 1465
d8 k create secret 1468
d8 k create secret docker-registry 1469
d8 k create secret generic 1471
d8 k create secret tls 1473
d8 k create service 1475
d8 k create service clusterip 1477
d8 k create service externalname 1478
d8 k create service loadbalancer 1480
d8 k create service nodeport 1482
d8 k create serviceaccount 1484
d8 k create token 1486
d8 k debug 1488
d8 k delete 1490
d8 k describe 1493
d8 k diff 1494
d8 k drain 1496
d8 k edit 1498
d8 k events 1500
d8 k exec 1502
d8 k explain 1504
d8 k expose 1505
d8 k get 1508
d8 k kustomize 1510
d8 k label 1512
d8 k logs 1515
d8 k options 1517
d8 k patch 1518
d8 k plugin 1520
d8 k plugin list 1521
d8 k port-forward 1523
d8 k proxy 1524
d8 k replace 1526
d8 k rollout 1528
d8 k rollout history 1530
d8 k rollout pause 1532
d8 k rollout restart 1533
d8 k rollout resume 1535
d8 k rollout status 1537
d8 k rollout undo 1539
d8 k run 1541
d8 k scale 1544
d8 k set 1546
d8 k set env 1547
d8 k set image 1549
d8 k set resources 1551
d8 k set selector 1554
d8 k set serviceaccount 1556
d8 k set subject 1558
d8 k taint 1560
d8 k top 1562
d8 k top node 1563
d8 k top pod 1565
d8 k uncordon 1566
d8 k version 1568
d8 k wait 1569
d8 mirror 1571
d8 mirror pull 1571
d8 mirror push 1575
d8 network 1575
d8 network cni-migration 1575

d8 network cni-migration cleanup 1575
d8 network cni-migration switch 1576
d8 network cni-migration watch 1576
d8 package 1576
d8 package bootstrap 1576
d8 package bootstrap application 1576
d8 package bootstrap module 1576
d8 package build 1577
d8 package render 1577
d8 package verify 1577
d8 snapshot 1578
d8 snapshot create 1578
d8 snapshot delete 1578
d8 snapshot describe 1579
d8 snapshot download 1579
d8 snapshot get 1580
d8 snapshot local 1580
d8 snapshot local describe 1580
d8 snapshot local get 1580
d8 snapshot restore 1581
d8 snapshot upload 1581
d8 status 1582
d8 stronghold 1582
d8 stronghold audit 1582
d8 stronghold audit disable 1582
d8 stronghold audit enable 1582
d8 stronghold audit list 1583
d8 stronghold auth 1583
d8 stronghold auth disable 1583
d8 stronghold auth enable 1583
d8 stronghold auth help 1583
d8 stronghold auth list 1583
d8 stronghold auth move 1583
d8 stronghold auth tune 1583
d8 stronghold debug 1583
d8 stronghold delete 1584
d8 stronghold events 1584
d8 stronghold events subscribe 1584
d8 stronghold kv 1584
d8 stronghold kv delete 1584
d8 stronghold kv destroy 1584
d8 stronghold kv enable-versioning 1584
d8 stronghold kv get 1584
d8 stronghold kv list 1585
d8 stronghold kv metadata 1585
d8 stronghold kv metadata delete 1585
d8 stronghold kv metadata get 1585
d8 stronghold kv metadata patch 1585
d8 stronghold kv metadata put 1585
d8 stronghold kv patch 1585
d8 stronghold kv put 1585
d8 stronghold kv rollback 1585
d8 stronghold kv undelete 1586
d8 stronghold lease 1586
d8 stronghold lease lookup 1586
d8 stronghold lease renew 1586
d8 stronghold lease revoke 1586
d8 stronghold list 1586
d8 stronghold login 1586
d8 stronghold monitor 1586
d8 stronghold namespace 1587
d8 stronghold namespace create 1587

d8 stronghold namespace delete 1587
d8 stronghold namespace list 1587
d8 stronghold namespace lock 1587
d8 stronghold namespace lookup 1587
d8 stronghold namespace patch 1587
d8 stronghold namespace unlock 1587
d8 stronghold operator 1587
d8 stronghold operator generate-root 1588
d8 stronghold operator init 1588
d8 stronghold operator key-status 1588
d8 stronghold operator members 1588
d8 stronghold operator raft 1588
d8 stronghold operator raft autopilot 1588
d8 stronghold operator raft autopilot get-config 1588
d8 stronghold operator raft autopilot set-config 1588
d8 stronghold operator raft autopilot state 1589
d8 stronghold operator raft demote 1589
d8 stronghold operator raft join 1589
d8 stronghold operator raft list-peers 1589
d8 stronghold operator raft promote 1589
d8 stronghold operator raft remove-peer 1589
d8 stronghold operator raft snapshot 1589
d8 stronghold operator raft snapshot restore 1589
d8 stronghold operator raft snapshot save 1589
d8 stronghold operator rekey 1590
d8 stronghold operator rotate 1590
d8 stronghold operator seal 1590
d8 stronghold operator step-down 1590
d8 stronghold operator unseal 1590
d8 stronghold operator usage 1590
d8 stronghold patch 1590
d8 stronghold path-help 1590
d8 stronghold pki 1591
d8 stronghold pki health-check 1591
d8 stronghold pki issue 1591
d8 stronghold pki list-intermediates 1591
d8 stronghold pki reissue 1591
d8 stronghold pki verify-sign 1591
d8 stronghold plugin 1591
d8 stronghold plugin deregister 1591
d8 stronghold plugin info 1591
d8 stronghold plugin list 1592
d8 stronghold plugin register 1592
d8 stronghold plugin reload 1592
d8 stronghold plugin reload-status 1592
d8 stronghold policy 1592
d8 stronghold policy delete 1592
d8 stronghold policy list 1592
d8 stronghold policy read 1592
d8 stronghold policy write 1593
d8 stronghold print 1593
d8 stronghold print token 1593
d8 stronghold read 1593
d8 stronghold secrets 1593
d8 stronghold secrets disable 1593
d8 stronghold secrets enable 1593
d8 stronghold secrets list 1593
d8 stronghold secrets move 1593
d8 stronghold secrets tune 1594
d8 stronghold ssh 1594
d8 stronghold status 1594
d8 stronghold token 1594

d8 stronghold token capabilities 1594
d8 stronghold token create 1594
d8 stronghold token lookup 1594
d8 stronghold token renew 1594
d8 stronghold token revoke 1595
d8 stronghold transit 1595
d8 stronghold transit import 1595
d8 stronghold transit import-version 1595
d8 stronghold unwrap 1595
d8 stronghold version 1595
d8 stronghold version-history 1595
d8 stronghold write 1595
d8 system 1595
d8 system collect-debug-info 1596
d8 system edit 1596
d8 system edit cluster-configuration 1596
d8 system edit provider-cluster-configuration 1597
d8 system edit static-cluster-configuration 1597
d8 system get 1597
d8 system get cluster-configuration 1598
d8 system get provider-cluster-configuration 1598
d8 system get static-cluster-configuration 1598
d8 system logs 1598
d8 system module 1599
d8 system module apply-now 1599
d8 system module approve 1599
d8 system module disable 1599
d8 system module enable 1600
d8 system module list 1600
d8 system module maintenance 1600
d8 system module maintenance disable 1600
d8 system module maintenance enable 1601
d8 system module snapshots 1601
d8 system module values 1601
d8 system package 1601
d8 system package scan 1602
d8 system queue 1602
d8 system queue list 1602
d8 system queue main 1603
d8 tools 1603
d8 tools far-convert 1603
d8 tools gostsum 1603
d8 tools imagedigest 1603
d8 tools imagedigest add 1603
d8 tools imagedigest calculate 1604
d8 tools imagedigest calculate-from-file 1604
d8 tools imagedigest validate 1604
d8 tools pki 1605
d8 tools pki certs 1605
d8 tools pki certs check 1605
d8 tools pki certs renew 1605
d8 tools pki certs renew all 1605
d8 tools sig-migrate 1606
d8 v 1606
d8 v ansible-inventory 1607
d8 v collect-debug-info 1608
d8 v console 1610
d8 v evict 1611
d8 v migrate 1612
d8 v port-forward 1613
d8 v restart 1615
d8 v scp 1616

- d8 v ssh 1617
- d8 v start 1619
- d8 v stop 1620
- d8 v vnc 1621

Frequently Asked Questions 1624
- Common 1624
  - How can I apply an update for a specific module immediately? 1624
  - How can I apply an update DKP immediately, bypassing update windows, canary releases, and manual update mode? 1624
  - What to do if, with VPN enabled, the container with the installer cannot access the network? 1624
  - How do I change the release channel for a module? 1625
  - How do I configure an external module to update to patch versions only? 1626
  - What should I do if the module image did not download and the module did not reinstall? 1627
- Cluster and Infrastructure 1628
  - What to do if you encounter problems updating Kubernetes components on cluster nodes, synchronizing nodes, or applying NodeGroup Configuration? 1628
  - What to do if problems adding a node to the cluster via Cluster API Provider Static are present? 1628
  - How to change the instance type for nodes with the CloudPermanent type? 1629
  - What to do if it takes a long time to switch to custom nodes in lower-priority groups? 1630
    - If you encounter issues with node ordering in a cluster with a single node group 1631
    - Example of the node ordering process with default settings 1631
  - What to do if the resources generated by an object are not created after adding the object? 1632
- Deckhouse 1633
  - How can I check DKP versions available in different release channels? 1633
  - How to check the job queue in Deckhouse? 1633
    - How to check the status of all Deckhouse task queues? 1633
    - How to view the status of the main task queue? 1633
  - What should I do if DKP is not receiving updates from the configured channel? 1634
    - Example DNS settings that may cause issues resolving the DKP image registry IP address 1634
  - Does Deckhouse support realtime (rt) and low-latency Linux kernels? 1634
  - How can I know when a new DKP version is available for the cluster? 1635
  - How can I receive information about upcoming updates in advance? 1635
  - How can I tell that the update was successful? 1635
    - If something goes wrong 1635
  - How can I tell that an update is in progress? 1636
  - What to do if there are problems updating DKP? 1636
    - Deckhouse Kubernetes Platform update fails, one or more Deckhouse pods are in an unworkable state 1636
    - DKP update is stuck in the Release is suspended status 1636
- IAM 1637
  - What to do if there are problems applying DexProvider settings? 1637
- Kubernetes and Scheduling 1637
  - What to do when the API server is overloaded? 1637
  - How to check the Kubernetes version in use? 1638
- Network 1638
  - Is the ingress-nginx module supported? Will it affect passing a PCI DSS audit? 1638

Release notes for Deckhouse Kubernetes Platform 1638
- Version 1.76 1638
  - Important 1638
  - Deckhouse subsystem 1638
  - Kubernetes & Scheduling subsystem 1639
  - IAM subsystem 1639
  - Security subsystem 1639
  - Cluster & Infrastructure subsystem 1640
  - Network subsystem 1640
- Version 1.75 1640
  - Important 1640
  - Deckhouse subsystem 1641
  - Kubernetes & Scheduling subsystem 1642
  - IAM subsystem 1642
  - Security subsystem 1642

Cluster & Infrastructure subsystem 1642
Network subsystem 1643
Version 1.74 1643
Important 1643
Major changes 1643
Security 1644
Network 1644
Version 1.73 1644
Important 1644
Major changes 1644
Security 1645
Network 1645
Version 1.72 1645
Important 1645
Major changes 1645
Security 1646
Version 1.71 1646
Important 1646
Major changes 1646
Security 1647
Network 1647
Version 1.70 1648
Important 1648
Major changes 1648
Security 1649
Version 1.69 1649
Important 1649
Major changes 1649
Security 1650
Version 1.68 1650
Important 1650
Major changes 1650
Security 1651
Mandatory component restart 1651
Embedded Deckhouse Kubernetes Platform modules 1651
The admission-policy-engine module 1652
How validation failure messages are displayed 1652
Pod validation when policies are modified or new ones are added 1652
Pod Security Standards 1655
Operational policies 1656
Security policies 1657
Modifying Kubernetes resources 1659
Third-party components 1659
Configuration 1659
Parameters 1661
Custom resources 1663
OperationPolicy 1663
SecurityPolicyException 1667
SecurityPolicy 1672
Custom Resources (by Gatekeeper) 1681
Mutation Custom Resources 1681
AssignMetadata 1681
Assign 1682
ModifySet 1682
AssignImage 1683
FAQ 1683
How do I configure alternative security policy management solutions? 1683
How do I configure policy selectors? 1683
spec.match.labelSelector – pod selection 1684
spec.match.namespaceSelector – namespace selection 1684
Using selectors together 1685
How do I extend Pod Security Standards policies? 1685

How to allow some Pod Security Standards policies without disabling whole list? 1687
What if there are multiple policies (operational or security) that are applied to the same object? 1689
Verification of image signatures 1690
How to block deleting a node without a label 1690
How to deny kubectl exec and kubectl attach to specific Pods? 1691
Built-in policy for heritage: deckhouse Pods 1691
Custom policy example 1692
The cert-manager module 1693
Features of the cert-manager module (with the changes made) 1693
Monitoring 1693
Module roles 1693
Third-party components 1693
Configuration 1693
Parameters 1695
Custom Resources 1700
Usage 1700
An example of provisioning a certificate 1700
Issuing a DNS wildcard certificate using Cloudflare 1701
Issuing a DNS wildcard certificate using Route53 1702
Issuing a DNS wildcard certificate using Google 1703
Issuing a self-signed certificate 1703
FAQ 1704
What types of certificates are supported? 1704
How to add an additional ClusterIssuer? 1704
When is an additional ClusterIssuer required? 1704
How to add an additional ClusterIssuer using Let’s Encrypt and DNS-01 verification method? 1705
How to add an additional Issuer and ClusterIssuer using HashiCorp Vault to issue certificates? 1705
How to add ClusterIssuer using its own or intermediate CA to to issue certificates? 1706
How to secure cert-manager credentials? 1707
Does the tls-acme annotation work? 1708
How do I check the certificate status? 1709
How do I get a list of certificates? 1710
The “CAA record does not match issuer” error 1710
The chrony module 1710
How it works 1710
Third-party components 1711
Configuration 1711
Parameters 1713
An example of the configuration 1713
FAQ 1713
How do I disable chrony and use NTP daemon on nodes? 1713
The cilium-hubble module 1714
Requirements 1714
Third-party components 1714
Configuration 1714
Requirements 1716
Conversions 1716
Parameters 1716
Authentication 1719
Cloud provider — AWS 1719
Configuration 1719
Parameters 1721
Provider configuration 1722
AWSClusterConfiguration 1722
Preparing environment 1730
JSON Policy 1730
Configuring IAM via the web interface 1733
Configuring IAM via the CLI 1733
Configuring IAM via Terraform 1734

Custom Resources 1735
AWSInstanceClass 1735
FAQ 1738
How do I create a peering connection between VPCs? 1738
How do I create a cluster in a new VPC with access over an existing bastion host? 1739
How do I create a cluster in a new VPC and set up bastion host to access the nodes? 1739
Configuring a bastion host 1739
Adding CloudStatic nodes to a cluster 1739
How to increase the size of a volume? 1740
How do I configure access to Amazon ECR repository on cluster nodes? 1740
What to do if switching to nodes in lower-priority groups takes a long time? 1740
Examples 1740
An example of the AWSInstanceClass custom resource 1740
LoadBalancer 1741
Service object Annotations 1741
Configuring security policies on nodes 1741
Enabling additional security groups on static and master nodes 1741
Enabling additional security groups on ephemeral nodes 1741
Configuring the load balancer if Ingress nodes are not available in all zones 1741
Layouts 1741
WithoutNAT 1742
WithNAT 1743
Cloud provider — Azure 1745
Configuration 1746
Parameters 1748
Provider configuration 1749
AzureClusterConfiguration 1749
Preparing environment 1756
Custom Resources 1757
AzureInstanceClass 1758
Examples 1761
An example of the AzureInstanceClass custom resource 1761
Layouts 1761
Standard 1761
Cloud provider — DVP 1762
Configuration 1762
Parameters 1764
Provider configuration 1765
DVPClusterConfiguration 1765
Preparing environment 1776
Creating a user 1776
Adding a role 1777
Generating a kubeconfig 1777
Custom Resources 1779
DVPInstanceClass 1779
Layouts 1781
Standard 1781
Cloud provider — Basis Dynamix 1782
Configuration 1783
Provider configuration 1784
DynamixClusterConfiguration 1784
Preparing environment 1789
Prepare an operating system image 1790
Custom Resources 1790
DynamixInstanceClass 1790
FAQ 1791
How to configure a LoadBalancer? 1791
Cloud provider - Basis Dynamix: Layouts 1791
Standard 1791
StandardWithInternalNetwork 1792
Cloud provider — GCP 1793
Configuration 1793

Parameters 1795
Provider configuration 1796
GCPClusterConfiguration 1796
Preparing environment 1801
Setup using Google Cloud Console 1801
Setup using gcloud CLI 1801
Custom Resources 1802
GCPInstanceClass 1802
FAQ 1807
How do I create a cluster? 1807
Adding CloudStatic nodes to a cluster 1807
Examples 1808
An example of the GCPInstanceClass custom resource 1808
Enabling nested virtualization 1808
Adding additional disks 1808
Configuring security policies on nodes 1808
Enabling additional network tags on static and master nodes 1808
Enabling additional network tags on ephemeral nodes 1809
Layouts 1809
Standard 1809
WithoutNAT 1810
Cloud provider — Huawei Cloud 1812
Configuration 1812
Provider configuration 1813
HuaweiCloudClusterConfiguration 1813
Preparing environment 1818
Configuring IAM via the web interface 1819
JSON policies 1819
Custom Resources 1821
HuaweiCloudInstanceClass 1821
Cloud provider - HuaweiCloud: Layouts 1823
Standard 1823
VpcPeering 1824
Cloud provider — OpenStack 1825
Configuration 1826
List of required OpenStack services 1827
Parameters 1828
Provider configuration 1831
OpenStackClusterConfiguration 1831
Preparing environment 1840
Custom Resources 1840
OpenStackInstanceClass 1840
FAQ 1844
How do I set up LoadBalancer? 1844
IngressNginxController example 1844
How do I set up security policies on cluster nodes? 1845
Enabling additional security groups on static and master nodes 1845
Enabling additional security groups on ephemeral nodes 1845
How do I create NodeGroups in availability zones? 1845
How do I create a hybrid cluster? 1846
Attaching storage devices to instances in a hybrid cluster 1846
How do I create an image in OpenStack? 1846
How to check whether the provider supports SecurityGroups? 1847
How to set up online disk resize 1848
How to use rootDiskSize and when it is preferred? 1848
Disks in OpenStack 1848
The rootDiskSize parameter 1848
Network disk is recommended for master nodes and bastion host 1848
Local disk is recommended for ephemeral nodes 1848
How do I check the disk volume in a flavor? 1848
How to override a default volume type of cloud provider? 1849
OFFLINE disk resize 1849

What to do if switching to nodes in lower-priority groups takes a long time? 1849
Examples 1849
Example 1 1849
Example 2 1849
Cloud provider - OpenStack: Layouts 1850
Standard 1850
StandardWithNoRouter 1852
Simple 1854
SimpleWithInternalNetwork 1856
Cloud provider — VMware Cloud Director 1858
Configuration 1859
Parameters 1860
Provider configuration 1861
VCDClusterConfiguration 1861
Preparing environment 1869
List of required VCD resources 1869
User permissions 1870
Adding a network 1873
Configuring DHCP 1876
Adding a vApp 1878
Adding a network to the vApp 1879
Incoming traffic 1880
Configuring DNAT/SNAT rules on the edge gateway 1880
Let’s Encrypt certificate issuance 1881
Configuring a firewall 1882
Virtual machine template 1884
Making a template from an OVA file 1885
Setting up the template in VCD 1889
Using the storage 1894
Using the LoadBalancer 1895
Custom Resources 1895
VCDInstanceClass 1895
Examples 1896
Example configuration of the VCDInstanceClass custom resource 1897
Cloud provider - VMware Cloud Director: Layouts 1897
Standard 1897
WithNAT 1898
Cloud provider — VMware vSphere 1900
Configuration 1900
Conversions 1902
Storage 1902
CSI 1902
Important information concerning the increase of the PVC size 1902
Parameters 1902
Provider configuration 1906
VsphereClusterConfiguration 1906
Preparing environment 1917
Environment requirements 1917
List of required vSphere resources 1918
List of required privileges 1918
vSphere configuration 1920
Configuration in vSphere Client 1920
Creating tags and tag categories in vSphere Client 1920
Configuring Datastore in vSphere Client 1923
Creating and assigning a role in vSphere Client 1924
Configuration via govc 1925
Installing govc 1925
Creating tags and tag categories with govc 1925
Datastore configuration with govc 1926
Creating and assigning a role with govc 1926
VM image requirements 1927
Preparing the virtual machine image 1927

Infrastructure 1928
Networking 1928
Inbound traffic 1928
Using the datastore 1929
Custom Resources 1929
VsphereInstanceClass 1929
Examples 1933
An example of the configuration 1933
An example of the VsphereInstanceClass custom resource 1933
Cloud provider - VMware vSphere: Layouts 1933
Standard 1933
Cloud provider — Yandex Cloud 1935
Configuration 1935
Storage 1937
LoadBalancer 1937
Parameters 1937
Provider configuration 1938
YandexClusterConfiguration 1938
Preparing environment 1947
Creating a service account 1947
Quotas 1948
Yandex Cloud integration 1948
Configuring security groups 1948
Yandex Lockbox integration 1949
Deployment instructions 1949
Checking that everything works as expected 1950
Yandex Managed Service for Prometheus integration 1951
Writing metrics 1951
Reading metrics with Grafana 1951
Custom Resources 1953
YandexInstanceClass 1953
FAQ 1957
Setting up the INTERNAL LoadBalancer 1957
Using a separate target group for a NodeGroup 1957
Reserving a public IP address 1958
dhcpOptions-related problems and ways to address them 1959
Editing parameters 1959
Aspects of the use 1959
Changing networkType does not recreate CloudEphemeral nodes 1959
Setting a custom StorageClass as default 1959
Resizing a PVC 1959
Adding CloudStatic nodes to a cluster 1962
Creating a cluster in a new VPC and setting up a bastion host to access the nodes 1962
Long switching to nodes in lower-priority groups 1963
Examples 1963
An example of the module configuration 1963
An example of the YandexInstanceClass custom resource 1963
LoadBalancer 1963
Service annotations 1963
Target group health checks 1964
Layouts 1964
Standard 1964
WithoutNAT 1967
WithNATInstance 1968
Cloud provider — zVirt 1970
Configuration 1971
Provider configuration 1972
ZvirtClusterConfiguration 1972
Preparing environment 1979
Environment requirements 1979
Prepare an operating system image 1979
Prepare a virtual machine template 1980

- Custom Resources 1983
  - ZvirtInstanceClass 1983
- FAQ 1985
  - How to get vNicProfileId 1985
- Cloud provider - zVirt: Layouts 1986
  - Standard 1986

The cni-cilium module 1987
- Limitations 1987
- Handling external traffic in different bpfLB modes (replacing kube-proxy from Cilium) 1987
- Using CiliumClusterwideNetworkPolicies 1989
- Changing Cilium Operation Mode 1989
- Disabling the kube-proxy Module 1989
- Using selective load balancing algorithm for services 1989
- Using Egress Gateway 1989
  - Basic mode 1989
  - Virtual IP mode 1990
- Exporting data from Hubble 1990
- Third-party components 1990
- Configuration 1990
  - Parameters 1992
- Custom Resources 1997
  - EgressGatewayPolicy 1997
  - EgressGateway 1999
  - HubbleMonitoringConfig 2001
- Examples 2013
  - Egress Gateway 2013
    - Operation principle 2013
    - Node maintenance 2014
    - Comparison with CiliumEgressGatewayPolicy 2014
    - Egress Gateway configuration examples 2014
      - EgressGateway in PrimaryIPFromEgressGatewayNodeInterface mode (basic mode) 2014
      - EgressGateway in VirtualIPAddress mode (Virtual IP mode) 2014
      - EgressGatewayPolicy 2015
  - HubbleMonitoringConfig 2015
    - HubbleMonitoringConfig configuration examples 2015
      - Enabling extended metrics and flow logs export (with filters and field mask) 2015
    - Collecting Hubble flow logs with the log-shipper module 2016

The cni-flannel module 2016
- Third-party components 2016
- Configuration 2016
  - Parameters 2018

The cni-simple-bridge module 2018

Managing control plane 2019
- Managing certificates 2019
- Scaling 2019
  - Scaling master nodes 2020
  - Dynamic terminated pod garbage collection threshold 2020
- Version control 2020
- Auditing 2021
- Feature Gates 2021
  - Kubernetes 1.31 2021
  - Kubernetes 1.32 2022
  - Kubernetes 1.33 2022
  - Kubernetes 1.34 2022
  - Kubernetes 1.35 2023
- Protecting sensitive fields in custom resources 2023
- Third-party components 2024
- Configuration 2024
  - Conversions 2026
  - Parameters 2026
- Custom Resources 2031
  - KubeSchedulerWebhookConfiguration 2031

FAQ 2033
How do I add a master node to a static or hybrid cluster? 2033
How do I add master nodes to a cloud cluster? 2034
How do I reduce the number of master nodes in a cloud cluster? 2035
How do I dismiss the master role while keeping the node? 2036
How do I switch to a different OS image in a multi-master cluster? 2037
In a cloud cluster 2037
In a static cluster 2038
How do I switch to a different OS image in a single-master cluster? 2040
In a cloud cluster 2040
In a static cluster 2040
How to configure HA mode with two master nodes and an arbiter node? 2040
Configuring in a cloud cluster 2041
Configuring in a static cluster 2042
How do I view the list of etcd members? 2043
Option 1 2043
Option 2 2043
What if something went wrong? 2043
What if the etcd cluster fails? 2043
What if etcd restarts with an error? 2044
What to do if the database volume of etcd reaches the limit set in quota-backend-bytes? 2044
How to compact etcd storage 2045
How to compact etcd storage on a node in a single-master cluster 2045
How to compact etcd storage in a cluster with multiple master nodes 2046
Cluster admin access model 2047
Kubeconfig files on master nodes 2047
RBAC-based admin access 2048
Recommended admin access 2048
Root kubeconfig symlink 2048
Security hardening 2048
Break-glass access 2048
How do I configure additional audit policies? 2049
How to omit Deckhouse built-in policy rules? 2049
How to stream audit log to stdout instead of files? 2049
How to deal with the audit log? 2050
How do I speed up the restart of Pods if the connection to the node has been lost? 2050
etcd backup and restore 2051
What is done automatically 2051
How to manually backup etcd 2051
Using Deckhouse CLI (Deckhouse Kubernetes Platform v1.65+) 2051
Using bash (Deckhouse Kubernetes Platform v1.64 and older) 2051
How do perform a full recovery of the cluster state from an etcd backup? 2051
Restoring a single-master cluster 2051
Restoring a multi-master cluster 2052
How do I restore a Kubernetes object from an etcd backup? 2053
Example of steps to restore objects from an etcd backup 2053
How the node to run the Pod on is selected 2055
The structure of the Kubernetes scheduler 2055
Working logic 2055
Scheduler profiles 2055
Pod scheduling stages 2056
Documentation 2056
How to change or extend the scheduler logic 2056
How does kubelet certificate rotation work? 2056
Certificates lifetime 2057
Specifics of working with kubelet server certificates in Deckhouse Kubernetes Platform 2057
How to manually update control plane component certificates? 2057
How do I protect sensitive fields in custom resources? 2057
Protection mechanisms 2058
Examples 2058
The connection of an external scheduler plugin 2058
Protecting resources with sensitive fields 2059

The csi-vsphere module 2060
Main Features 2061
System requirements 2061
Architecture 2061
Compatibility modes 2061
Storage discovery and provisioning 2061
Default StorageClass and exclusions 2062
Volume lifecycle 2062
Limitations 2062
Configuration 2062
Parameters 2064
Preparing environment 2065
Required resources 2065
Installing govc 2066
Create tags and tag categories 2066
Datastore configuration 2066
Create and assign the role 2066
The deckhouse module 2067
Deckhouse releases update 2067
Get Deckhouse releases status 2067
Update process 2068
Manual release deployment 2068
Pin a release 2068
Priority Classes 2068
Third-party components 2069
Configuration 2070
Parameters 2071
Custom Resources 2083
ConversionWebhook 2083
ValidationWebhook 2086
Usage 2092
Setting up the update mode 2092
Update windows configuration 2092
Manual update confirmation 2093
Deckhouse update notifications 2093
Reserving the hostnames of the platform web interfaces 2095
Reserved hostnames 2096
Excluding a name from reservation 2096
Excluding the shared Gateway 2096
Reservation with existing applications in the cluster 2097
Checking reservation status 2097
Collect debug info 2098
FAQ 2098
How to run kube-bench in my cluster? 2098
How to collect debug info? 2098
How to debug pod problems with ephemeral containers? 2100
How to debug node problems with ephemeral containers? 2100
The deckhouse-tools module 2100
Configuration 2100
An example of the configuration 2100
Parameters 2102
The descheduler module 2105
Features of the module 2105
Metrics provider 2105
Strategies 2105
HighNodeUtilization 2105
LowNodeUtilization 2106
RemoveDuplicates 2106
RemovePodsHavingTooManyRestarts 2106
RemovePodsViolatingInterPodAntiAffinity 2106
RemovePodsViolatingNodeAffinity 2107
RemovePodsViolatingTopologySpreadConstraint 2107

- Third-party components 2107
- Configuration 2107
  - Conversions 2109
  - Parameters 2109
- Custom Resources 2110
- Descheduler 2110
- Examples 2118
- Configuring the pod redistribution interval 2118
- Example LowNodeUtilization strategy 2118
- Example HighNodeUtilization strategy 2118
- The documentation module 2119
- Configuration 2119
  - Parameters 2120
  - Authentication 2122
- Examples 2123
  - An example of the module configuration 2123
- The extended-monitoring module 2123
- Configuration 2123
  - How to use extended-monitoring-exporter 2123
    - Standard labels and supported Kubernetes objects 2124
      - Non-namespaced Kubernetes objects 2124
      - Node 2124
      - Namespaced Kubernetes objects 2124
      - Pod 2124
      - Ingress 2124
      - Deployment 2124
      - StatefulSet 2124
      - DaemonSet 2124
      - CronJob 2125
    - How does it work? 2125
  - Requirements 2126
  - Conversions 2126
  - Parameters 2126
- The extended monitoring module: FAQ 2128
  - How to switch to HTTP instead of HTTPS checks of my registry? 2128
- The ingress-nginx module 2128
- Traffic routing 2129
- Supported Protocols 2129
- Using HTTPS 2129
- Monitoring and statistics 2129
- Statistics 2129
  - Basic principles of collecting statistics 2129
  - What information does Prometheus collect, and in what form? 2129
- Module support and security 2130
- Third-party components 2130
- Configuration 2130
  - Parameters 2132
- Custom Resources 2132
  - IngressNginxController 2132
- FAQ 2151
  - How do I limit access to the application in the cluster to Ingress controllers only? 2151
    - How do I provide minimum rights to the Service Account? 2152
    - The Kube-RBAC-Proxy configuration 2152
  - How do I configure an external load balancer to check if IngressNginxController is available? 2153
  - How do I configure MetalLB to be accessible from the internal network only? 2153
  - How to add extra log fields to a nginx-controller? 2154
  - How to enable HorizontalPodAutoscaling for IngressNginxController? 2154
  - How to use IngressClass with IngressClassParameters? 2154
  - How to disable the collection of detailed Ingress resources statistics? 2154
  - How do I correctly drain a node running an IngressNginxController’s pods? 2155
  - How to enable Web Application Firewall (WAF)? 2155

Enabling ModSecurity 2155
Setting up ModSecurity 2155
Adding SecRules 2155
Enabling OWASP Core Rule Set (CRS) 2156
Examples 2156
An example for AWS (Network Load Balancer) 2156
An example for GCP / Yandex Cloud / Azure 2156
An example for OpenStack 2156
An example for Bare metal 2157
An example for Bare metal (Behind external load balancer, e.g. Cloudflare, Qrator, Nginx+, Citrix ADC, Kemp, etc.) 2157
An example for Bare metal (MetalLB BGP LoadBalancer) 2157
An example for Bare metal (MetalLB L2 LoadBalancer) 2157
Example of segregating access between public and administrative zones 2158
Specifics of using a single Ingress controller 2158
Using separate Ingress controllers 2159
The istio module 2160
Compatibility table for supported versions 2161
What issues does Istio help to resolve? 2161
Mutual TLS 2161
Authorization 2161
Request routing 2161
Managing request balancing between service Endpoints 2161
Observability 2162
Tracing 2162
Grafana 2162
Kiali 2162
Architecture of the cluster with Istio enabled 2162
Application service architecture with Istio enabled 2163
Details 2163
User request lifecycle 2163
Application with Istio turned off 2163
Application with Istio turned on 2163
Activating Istio to work with the application 2163
Federation and multicluster 2164
Federation 2164
Requirements for clusters 2164
General principles of federation 2164
Enabling the federation 2164
Managing the federation 2164
Multicluster 2165
Requirements for clusters 2165
General principles 2165
Enabling the multicluster 2165
Managing the multicluster 2165
Estimating overhead 2165
control-plane 2165
data-plane 2165
Third-party components 2166
Configuration 2166
Conversions 2167
Parameters 2167
Authentication 2182
Custom Resources 2182
IngressIstioController 2182
IstioFederation 2185
IstioMulticluster 2186
Custom Resources (by istio.io) 2187
Routing 2187
DestinationRule 2187
VirtualService 2187
ServiceEntry 2187

Authentication 2187
PeerAuthentication 2187
RequestAuthentication 2187
Authorization 2187
AuthorizationPolicy 2188
Sidecar 2188
Examples 2188
Circuit Breaker 2188
gRPC balancing 2189
Locality Failover 2189
Retry 2189
Canary 2189
Cookie-based routing 2190
Probability-based routing 2190
Ingress to publish applications 2190
Istio Ingress Gateway 2190
Ingress NGINX 2192
Authorization configuration examples 2192
Decision-making algorithm 2192
Deny all actions for the foo namespace 2193
Deny requests from the foo NS only 2194
Allow requests for the foo NS only 2194
Allow requests from anywhere in the cluster 2194
Allow any requests for foo or bar clusters 2194
Allow any requests only from entities in the baz namespace of the foo or bar clusters 2194
Allow from any cluster (via mTLS) 2195
Allow all requests from anywhere (including no mTLS - plain text traffic) 2195
Setting up federation for two clusters using the IstioFederation CR 2195
Setting up multicluster for two clusters using the IstioMulticluster CR 2195
Control the data-plane behavior 2196
Prevent istio-proxy from terminating before the main application’s connections are closed 2196
CNIPlugin application traffic redirection mode restrictions 2196
Upgrading Istio 2196
Upgrading Istio control-plane 2196
Auto upgrading istio data-plane 2197
Customizing istio-proxy sidecar resource management 2197
Supported annotations 2197
Configuration Examples 2197
The keepalived module 2198
Third-party components 2198
Configuration 2198
Custom Resources 2199
KeepalivedInstance 2199
keepalived: FAQ 2201
How to manually switch keepalived? 2201
Examples 2201
Three public IP addresses 2201
The kube-dns module 2202
Third-party components 2202
Configuration 2203
Parameters 2204
FAQ 2205
How to change cluster domain with minimal downtime? 2205
How to scale-up kube-dns replicas? 2206
Examples 2206
Configuration example 2207
The kube-proxy module 2207
The local-path-provisioner module 2207
How does it work? 2207
Limitations 2208
Third-party components 2208
Configuration 2208

Custom Resources 2209
LocalPathProvisioner 2209
FAQ 2210
How to configure Prometheus to use local storage for storing data? 2210
Examples 2210
Example of a LocalPathProvisioner custom resource 2210
Example of a LocalPathProvisioner custom resource with reclaimPolicy set 2210
The log-shipper module 2210
Deployment topologies 2211
Distributed 2211
Centralized 2211
Stream 2211
Metadata 2211
Kubernetes 2211
File 2211
Log filters 2212
Third-party components 2212
Configuration 2212
Conversions 2213
Parameters 2213
Custom Resources 2216
ClusterLogDestination 2216
ClusterLoggingConfig 2244
PodLoggingConfig 2253
FAQ 2256
How to add authorization to the ClusterLogDestination resource? 2256
Advanced usage 2257
How to enable debugging logs? 2257
How to get aware of logs pipelines? 2257
See pipelines as a graph 2257
Investigate data processing 2258
Get raw log samples 2258
How to add a new source/sink support for log-shipper? 2258
Examples 2258
Getting logs from all cluster Pods and sending them to Loki 2258
Reading Pod logs from a specified namespace with a specified label and redirecting to Loki and Elasticsearch 2259
Creating a source in namespace and reading logs of all Pods in that NS with forwarding them to Loki 2259
Reading only Pods in the specified namespace and having a certain label 2259
Migration from Promtail to Log-Shipper 2260
Working with Grafana Cloud 2260
Adding Loki source to Deckhouse Grafana 2260
Elasticsearch < 6.X usage 2261
Index template for Elasticsearch 2261
Splunk integration 2261
Simple Logstash example 2262
Syslog 2262
Graylog integration 2262
Logs in CEF Format 2263
Collect Kubernetes Events 2263
Log filters 2264
Collect only logs of the nginx container 2264
Collect logs without strings GET /status" 200 2264
Audit of kubelet actions 2264
Deckhouse system logs 2265
Log transformations 2265
Transforming logs into a structured object 2265
Transforming logs in Klog format into a structured object 2266
Transforming logs in Syslog format into a structured object 2266
Transforming logs in CLF format into a structured object 2267
Transforming logs in logfmt format into a structured object 2267

Parsing JSON and reducing the nesting depth 2268
Example of parsing mixed-format entries into a structured object 2268
Replacing labels 2269
Replacing substrings in string fields 2269
Adding labels from literals and templates 2270
Removing labels 2270
Example of removing a specific label from a structured message 2271
Sending logs to Elasticsearch with transformations 2271
Collect logs from production namespaces using the namespace label selector option 2272
Exclude Pods or namespaces with a label 2272
Enable Buffering 2273
Example of enabling in-memory buffering 2273
Example of enabling disk buffering 2273
Example of defining behavior when the buffer is full 2273
The loki module 2274
Third-party components 2274
Configuration 2274
Parameters 2275
An example of the configuration 2280
FAQ 2280
Why is LokiDiscardedSamplesWarning alert triggered? 2280
Possible causes 2280
Resolving the issue 2280
Module loki: examples 2280
Reading Pod logs from a specified namespace and sending them to Loki 2280
The metallb module 2281
Layer 2 mode 2281
Advantages of the module over the classic MetalLB 2281
BGP Mode 2282
How metallb Works in BGP Mode 2282
Advantages of Using BGP 2282
Disadvantages of Using BGP 2282
Third-party components 2282
Configuration 2282
Conversions 2284
Parameters 2284
Custom Resources 2287
MetalLoadBalancerClass 2287
Examples 2288
Example of metallb usage in L2 LoadBalancer mode 2288
Example of metallb usage in BGP LoadBalancer mode 2289
Additional configuration examples for Service 2290
Creating a service and assigning it specific IP addresses from the pool 2290
Creating a service with a single, forcibly selected address 2290
Creating services with shared IP addresses 2291
Creating a service and assigning IPAddressPools to it when using mettalb in BGP LoadBalancer mode 2291
Automatic configuration of the monitoring system to collect metrics from user applications 2292
An example: Service 2293
An example: Deployment 2293
Configuration 2293
Requirements 2295
The monitoring-deckhouse module 2295
Metrics collection 2295
Integration with Observability module 2295
Requirements 2296
Configuration 2296
Requirements 2297
The monitoring-kubernetes module 2297
Third-party components 2298
Configuration 2298
Requirements 2299

Conversions 2299
Parameters 2299
Monitoring the control plane 2301
Configuration 2301
Requirements 2302
Monitoring network interaction between all cluster nodes, as well as (optionally) up to additional external nodes 2303
Description 2303
How does it work? 2303
The monitoring-ping module: configuration 2303
Requirements 2304
Parameters 2304
The monitoring-ping module: usage 2305
Adding extra IP addresses to monitor 2305
Module multitenancy-manager 2305
Description 2305
Why is this needed? 2305
Advantages of the module 2306
Limitations 2306
Internal Logic 2306
Creating a project 2306
Isolating a project 2306
The multitenancy-manager module: configuration 2306
Parameters 2308
The multitenancy-manager module: Custom Resources 2308
Project 2308
ProjectTemplate 2309
The multitenancy-manager module: usage examples 2310
Default project templates 2310
Creating a project 2311
Creating a project automatically for a namespace 2311
Creating your own project template 2312
Using labels to manage resources 2312
Skipping creation of the heritage: multitenancy-manager label 2312
Excluding resources from management by multitenancy-manager 2312
Implementing validation of object changes with a custom label 2313
How validation works in multitenancy-manager 2313
Creating your own validation 2313
The namespace-configurator module 2314
How does it work? 2314
What do I need to configure? 2314
Configuration 2314
Parameters 2316
Examples 2316
Example 2316
The network-gateway module 2317
Configuration 2317
Parameters 2318
The network-policy-engine module 2320
Third-party components 2320
Configuration 2320
Examples 2322
Examples of network policies 2322
Deny external traffic to Pods in the namespace, but allow traffic to external resources from within the namespace 2322
The node-local-dns module 2322
Module Description 2322
Purpose 2322
Grafana dashboard 2322
How does it work? 2323
Aspects of CoreDNS configuration 2323
Configuration 2323

Requirements 2324
Parameters 2324
Examples 2325
An example of configuring a custom DNS for a Pod 2325
Configuration example 2325
Managing nodes 2325
Primary functions 2326
Node types 2326
Node grouping and group management 2327
Delaying node reboot or shutdown while critical pods are running 2327
How it works and key features 2327
Delaying mechanism features in DKP 2327
Automatic deploying, configuring and updating Kubernetes nodes 2327
Deploying Kubernetes nodes 2327
Keeping nodes up-to-date 2327
Working with nodes on supported cloud platforms 2327
Scaling nodes in the cloud 2328
Working with static nodes 2328
Cluster API Provider Static 2328
Custom node settings 2329
Features of writing scripts 2331
Chaos Monkey 2332
Monitoring 2332
What information does Prometheus collect and in what form? 2332
Third-party components 2332
Configuration 2332
Conversions 2334
Parameters 2334
Custom resources 2335
Instance 2335
InstanceTypesCatalog 2336
NodeGroup 2337
NodeGroupConfiguration 2369
NodeUser 2371
SSHCredentials 2374
StaticInstance 2376
FAQ 2383
How do I add master nodes to a cloud cluster? 2383
How do I reduce the number of master nodes in a cloud cluster? 2383
Static nodes 2383
How do I add a static node to a cluster (Cluster API Provider Static)? 2383
How do I add a batch of static nodes to a cluster manually? 2384
How do I clean up a static node manually? 2385
For all nodes except the control-plane 2385
For control-plane nodes 2386
Can I delete a StaticInstance? 2386
How do I change the IP address of a StaticInstance? 2387
How do I migrate a manually configured static node under CAPS control? 2387
How do I change the NodeGroup of a static node? 2387
How to clean up a node for adding to the cluster? 2387
How do I know if something went wrong? 2387
How do I know what is running on a node while it is being created? 2388
How do I update kernel on nodes? 2388
Debian-based distros 2388
CentOS-based distros 2389
NodeGroup parameters and their result 2390
How do I redeploy ephemeral machines in the cloud with a new configuration? 2391
How do I allocate nodes to specific loads? 2391
How to allocate nodes to system components? 2391
Frontend 2391
System components 2391
How do I speed up node provisioning on the cloud when scaling applications horizontally? 2392

How do I disable machine-controller-manager/CAPI in the case of potentially cluster-damaging changes? 2392
How do I restore the master node if kubelet cannot load the control plane components? 2392
containerd 2392
How to change CRI for NodeGroup? 2393
Why might the CRI change not apply? 2393
How to change CRI for the whole cluster? 2394
How to add node configuration step? 2395
How to automatically put custom labels on the node? 2395
How to deploy custom containerd configuration? 2395
How to add configuration for an additional registry? 2397
Old Method 2397
How to add additional registry auth (deprecated method)? 2398
How to configure a certificate for an additional registry (deprecated method)? 2399
How to add TLS skip verify (deprecated method)? 2400
How to set up a mirror for public image registries (deprecated method)? 2401
New Method 2402
How to add additional registry auth (actual method)? 2403
How to configure a certificate for an additional registry (actual method)? 2403
How to add TLS skip verify (actual method)? 2404
How to set up a mirror for public image registries (actual method)? 2405
How to use NodeGroup’s priority feature 2406
How to interpret Node Group states? 2407
How do I make werf ignore the Ready conditions in a node group? 2409
What is an Instance resource? 2409
When is a node reboot required? 2409
How to enable a delay before a node shutdown or restart while critical pods are running on it? 2409
Searching pods that prevent shutdown and performing the node troubleshooting 2410
How the fencing mechanism handles different node types? 2410
Cloud nodes (CloudEphemeral, CloudPermanent) 2410
Static nodes (Static, CloudStatic) 2411
Disabling automatic reboot after kernel panic 2411
How do I work with GPU nodes? 2411
Step-by-step procedure for adding a GPU node to the cluster 2411
How to monitor GPUs? 2416
Which GPU modes are supported? 2416
How to view available MIG profiles in the cluster? 2416
How to set a custom per-GPU MIG layout on a node? 2416
MIG profile does not activate — what to check? 2417
Are AMD or Intel GPUs supported? 2417
Examples 2417
Examples of the NodeGroup configuration 2417
Cloud nodes 2417
Static nodes 2418
System nodes 2418
Nodes with GPU 2419
Exclusive mode (one Pod — one GPU) 2419
Time-slicing (4 partitions) 2420
MIG (all-1g.5gb profile) 2420
Smoke-test Job (CUDA vectoradd) 2420
Adding a static node to a cluster 2421
Manually 2421
Using the Cluster API Provider Static 2421
Using Cluster API Provider Static for multiple node groups 2423
Cluster API Provider Static: Moving Instances Between Node Groups 2425
Initial Configuration 2425
Steps to Move an Instance Between Node Groups 2426
1. Create a New NodeGroup for the Target Node Group 2426
2. Update the Label on the StaticInstance 2426
3. Decrease the Number of Static Instances in the Original NodeGroup 2426
An example of the NodeUser configuration 2427

- An example of the NodeGroupConfiguration configuration 2427
  - Installing the cert-manager plugin for kubectl on master nodes 2427
  - Tuning sysctl parameters 2427
  - Adding a root certificate to the host 2427
  - Adding the ability to download images from insecure container registry to containerd 2430
- The okmeter module 2430
  - Configuration 2430
    - Parameters 2431
  - Examples 2432
    - Example 2432
- The openvpn module 2433
  - Exposing the VPN service 2433
  - Available cluster resources after connecting to the VPN 2433
  - User’s traffic audit 2433
  - Third-party components 2433
  - Configuration 2434
    - Conversions 2435
    - Parameters 2435
    - Authentication 2439
  - FAQ 2440
    - Why doesn’t automatic DNS server configuration work when connecting to macOS and Linux using the OpenVPN client 2440
    - How to revoke, rotate or delete a user certificate 2440
    - How to rotate a server certificate 2440
    - How to rotate a root certificate (CA) 2441
  - Examples 2441
    - An example for bare metal clusters 2441
    - An example for AWS & Google Cloud 2441
    - An example for an external load balancer with a public IP address 2441
- The operator-prometheus module 2442
  - Prometheus 2442
    - Prometheus functions 2442
    - Prometheus configuration 2442
    - Sources of monitoring targets 2442
  - Prometheus Operator 2445
    - Prometheus Operator functions 2445
    - Contents of the Prometheus Pod 2445
    - ServiceMonitor processing 2445
    - Processing custom resources with rules 2446
  - Third-party components 2446
  - Configuration 2447
    - Parameters 2448
  - Custom resources 2449
    - PodMonitor 2449
    - Probe 2458
    - ScrapeConfig 2467
    - ServiceMonitor 2483
- The Prometheus monitoring module 2492
  - Monitoring hardware resources 2492
  - Kubernetes monitoring 2493
  - Ingress monitoring 2493
  - Advanced monitoring mode 2493
    - Alerting in advanced monitoring mode 2493
  - Monitoring Management as code (IaC approach) 2493
  - Alerts 2493
    - Sending alerts to external systems 2493
  - Architecture 2493
    - Basic monitoring components 2494
  - Fault Tolerance and High Availability Monitoring (HA) mode 2495
  - Third-party components 2495
- Configuration 2496
  - Requirements 2497

- Conversions 2497
- Parameters 2497
- Authentication 2502
- Notes 2502

Custom resources 2502
- ClusterAlert 2502
- CustomAlertmanager 2503
- CustomPrometheusRules 2533
- GrafanaAdditionalDatasource 2534
- GrafanaAlertsChannel 2535
- GrafanaDashboardDefinition 2536
- PrometheusRemoteWrite 2536

Usage 2539
- An example of the module configuration 2539
- Writing Prometheus data to the longterm storage 2539
  - Example of the basic PrometheusRemoteWrite 2539
  - Example of the expanded PrometheusRemoteWrite 2540
- Connecting Prometheus to an external Grafana instance 2540
- Connecting an external app to Prometheus 2541
- Sending alerts to Telegram 2542
- Example of sending alerts to Slack with a filter 2543
- Example of sending alerts to Opsgenie 2543
- Example of sending alert by Email 2544

FAQ 2544
- How do I collect metrics from applications running outside of the cluster? 2544
  - An example 2545
- How do I create custom Grafana dashboards? 2545
- How do I add alerts and/or recording rules? 2545
  - How do I provision additional Grafana data sources? 2546
- How do I enable secure access to metrics? 2546
  - An example of collecting metrics securely from an application inside a cluster 2546
  - An example of collecting metrics securely from an application outside a cluster 2548
- How do I add Alertmanager? 2549
- How do I add an additional Alertmanager? 2549
- How do I ignore unnecessary alerts in Alertmanager? 2549
- Why can’t different scrape Intervals be set for individual targets? 2550
- How do I limit Prometheus resource consumption? 2550
- How do I set up a ServiceMonitor or PodMonitor to work with Prometheus? 2550
- How do I set up a Probe to work with Prometheus? 2551
- How do I set up a PrometheusRules to work with Prometheus? 2551
- How to expand disk size 2551
- How to get information about alerts in a cluster? 2551
- How do I add additional endpoints to a scrape config? 2552

The prometheus-metrics-adapter module 2553
- How does it work? 2553
- Third-party components 2553
- Configuration 2553
  - Requirements 2554
  - Parameters 2554
- Custom resources 2555
  - ClusterDaemonSetMetric 2555
  - ClusterDeploymentMetric 2555
  - ClusterIngressMetric 2555
  - ClusterPodMetric 2556
  - ClusterServiceMetric 2556
  - ClusterStatefulSetMetric 2557
  - DaemonSetMetric 2557
  - DeploymentMetric 2557
  - IngressMetric 2558
  - NamespaceMetric 2558
  - PodMetric 2558
  - ServiceMetric 2559

StatefulSetMetric 2559
Usage 2559
Scaling types 2560
Classic resource consumption-based scaling 2560
Scaling based on custom metrics 2561
Registering custom metrics with Kubernetes API 2561
Using custom metrics in HPA 2561
Using custom metrics with the RabbitMQ queue size 2562
Using volatile custom metrics 2563
Examples involving custom metrics of type Pods 2563
Registering external metrics with the Kubernetes API 2564
Using external metrics in HPA 2564
Using the queue size in Amazon SQS 2565
Debugging 2566
How do I get a list of custom metrics? 2566
How do I get the value of a metric associated with an object? 2566
How do I get the value of a metric created via NamespaceMetric? 2566
How do I get external metrics? 2566
The Prometheus Pushgateway module 2566
Third-party components 2566
Configuration 2567
Requirements 2568
Parameters 2568
Examples 2568
Example of the module configuration 2568
Pushing a metric 2569
Deleting metrics 2569
Registry Module 2569
Description 2569
Restrictions and features of using the module 2569
Cluster installation limitations 2569
Operating conditions restrictions 2569
Mode switching restrictions 2570
Direct Mode Architecture 2570
Proxy Mode Architecture 2570
Local Mode Architecture 2570
Module registry: configuration 2571
Parameters 2572
FAQ 2573
How to Migrate to the registry module? 2573
For containerd v2 2573
For Containerd v1 2574
How to Migrate Back from the Registry Module? 2576
How to check the registry mode switch status? 2578
Module registry: usage example 2579
Switching to the Direct Mode 2579
Switching to the Proxy Mode 2580
Switching to the Local Mode 2581
Switching to the Unmanaged Mode 2583
The registry-packages-proxy module 2584
Architecture 2584
HTTP API 2584
Package icons (/v1/packages/) — in-cluster only 2584
Deckhouse CLI downloads (/v1/images/) — public 2585
Internal /package endpoint 2585
Package retrieval flow 2585
High availability 2585
RBAC roles provided by the module 2585
Limitations 2585
The registry-packages-proxy: configuration 2585
The secret-copier module 2587
How does it work? 2587

- What do I need to configure? 2587
- How to synchronize Secret to some selected namespaces instead of all namespaces? 2587
- The service-with-healthchecks module 2587
  - Limitations of the built-in Service load balancer 2587
  - ServiceWithHealthcheck load balancer capabilities 2588
  - How the ServiceWithHealthcheck load balancer works 2588
  - Configuration 2588
    - Parameters 2590
  - The service-with-healthchecks: Custom Resources 2590
    - ServiceWithHealthchecks 2590
  - Module service-with-healthchecks: examples 2596
    - Running two independent balancers on the same virtual machine 2597
      - Creating a virtual machine 2597
      - Web service and SMTP load balancer manifests 2597
    - Load balancers for working with a PostgreSQL cluster 2598
      - Creating a StatefulSet for PostgreSQL 2598
      - Configuring ServiceWithHealthchecks load balancers 2599
- The terraform-manager module 2599
  - Third-party components 2600
  - Configuration 2600
    - Parameters 2601
- Cluster SLA Monitoring 2602
  - Interface 2602
- Configuration 2603
  - Conversions 2604
  - Parameters 2605
  - Authentication 2611
- Custom Resources 2611
  - Downtime 2611
  - UpmeterHookProbe 2612
  - UpmeterRemoteWrite 2612
- FAQ 2615
  - Why can’t upmeter-probe-* pods be placed in the cluster, why are some pods constantly deleted? 2615
- Examples 2615
  - An example of the UpmeterRemoteWrite configuration 2615
- The user-authn module 2616
  - Integration features 2616
    - Basic authentication for the Kubernetes API requests 2616
    - Application integration 2616
    - Web interface for generating ready-made kubeconfig files 2616
  - Exposing the Kubernetes API over Ingress 2617
  - Extensions by Flant 2617
  - High availability mode 2617
  - Third-party components 2617
  - Configuration 2617
    - Conversions 2619
    - Parameters 2619
  - Custom Resources 2625
    - DexAuthenticator 2626
    - DexClient 2641
    - DexProvider 2643
    - Group 2665
    - UserOperation 2666
    - User 2668
  - Usage 2672
    - An example of the module configuration 2672
    - Configuring a provider 2673
      - Checking provider connectivity 2673
      - GitHub 2673
      - GitLab 2673
      - Atlassian Crowd 2674

Bitbucket Cloud 2674
OIDC (OpenID Connect) 2675
Keycloak 2675
Okta 2676
Blitz Identity Provider 2677
LDAP 2678
Configuring Basic Authentication 2678
Kerberos (SPNEGO) SSO for LDAP 2679
SAML 2680
Configuring the OAuth2 client in Dex for connecting an application 2681
Granting an application access to the Kubernetes API 2682
Local Authentication 2682
Creating a user 2682
Local user operations 2683
Administrative operations 2683
Self-service password reset 2684
Adding a user to a group 2684
Password policy 2685
Two-factor authentication (2FA) 2685
Assigning permissions to a user or group 2686
FAQ 2686
How to secure my application? 2686
Setting up CIDR-based restrictions 2688
Authentication flow with DexAuthenticator 2688
How to generate a kubeconfig and access Kubernetes API? 2688
Configuring kube-apiserver 2689
The flow of accessing Kubernetes API with generated kubeconfig 2689
How to rotate the secret of the kubernetes OAuth2 client? 2689
How to enable Kerberos (SPNEGO) SSO for LDAP? 2690
How to configure Basic Authentication for accessing Kubernetes API via LDAP? 2690
How is Dex protected against credential brute-forcing? 2690
The user-authz module 2690
Experimental role-based model 2691
Use roles 2691
Manage roles 2692
Subsystems of the role-based model 2693
Current role-based model 2694
Default access list for each role 2695
Configuration 2698
Parameters 2700
Custom Resources 2700
AuthorizationRule 2701
ClusterAuthorizationRule 2702
Usage 2709
Example of assigning rights to a cluster administrator 2709
Example of assigning rights to a network administrator 2709
Example of assigning administrative rights to a user within a namespace 2711
An example of ClusterAuthorizationRule 2712
Example of granting access to all namespaces 2713
Creating a user 2714
Creating a ServiceAccount for a machine and granting it access 2714
How to create a user using a client certificate 2716
Creating a user using a certificate issued via OpenSSL 2717
Creating a user by issuing a certificate via the Kubernetes API 2718
Granting access to the created user 2720
Configuring kube-apiserver for multi-tenancy mode 2720
How do I check that a user has access? 2721
Customizing rights of high-level roles 2722
FAQ 2723
How do I create a user? 2723
How do I limit user rights to specific namespaces? 2723
What if there are two ClusterAuthorizationRules matching to a single user? 2723

How do I extend a role or create a new one? 2724
Creating a new role subsystem 2724
Extending the custom role 2726
Extending the existing manage subsystem roles 2727
Extending manage subsystem roles and adding a new namespace 2727
Extending the existing use roles 2728
The vertical-pod-autoscaler module 2728
Grafana dashboard 2729
Vertical Pod Autoscaler Architecture 2729
Third-party components 2729
The vertical-pod-autoscaler: configuration 2729
Requirements 2731
Parameters 2731
Custom resources 2732
VerticalPodAutoscaler 2732
VerticalPodAutoscalerCheckpoint 2742
FAQ 2745
How do I view the Vertical Pod Autoscaler recommendations? 2745
How does Vertical Pod Autoscaler handle limits? 2745
Example No. 1 2746
Example No. 2 2746
Examples 2747
The module configuration 2747
The basic VerticalPodAutoscaler custom resource example 2747
The advanced VerticalPodAutoscaler custom resource example 2747

Deckhouse Kubernetes Platform documentation

Download PDF

Welcome to the home page of the Deckhouse Kubernetes Platform documentation. We recommend starting with the Getting started guide for those who haven't yet tried the platform. It provides step-by-step instructions for deploying the platform to any infrastructure.

Here are some tips on how to find what you need:

- If you know what you're looking for exactly, use the search.
- Check this list if you are looking for information on a particular module.
- The menu on the left is for searching by scope.

Note that the Deckhouse Kubernetes Platform documentation for different versions may vary. Select the version you need from the drop-down list at the top of the page. The list contains up-to-date versions of the documentation.

If you have any questions, feel free to contact us at Telegram channel. We will do our best to help you.

Commercial Edition users can contact us at email. We will be glad to provide assistance!

Interested in improving Deckhouse Kubernetes Platform? Feel free to open a GitHub Issue, suggest an idea or join the list of contributors.

Feel like you can contribute a lot more? Then join our team! We look forward to welcoming new professionals!

Installation

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

This page provides an overview of installing Deckhouse Kubernetes Platform (DKP).

Installation methods

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

Installation options

You can install DKP in the following ways::

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

Installation requirements

To estimate the resources required for Deckhouse Kubernetes Platform installation, refer to the following guides:

- Bare-metal cluster sizing guide
- Disk layout and sizing guide
- Production preparation guide

Before installation, ensure the following:

- For bare-metal clusters (including hybrid clusters) and installations in unsupported clouds: the server runs an OS from the supported OS list (or a compatible version) and is accessible via SSH with a key.
- For supported clouds: the required resource quotas are available and access credentials to the cloud infrastructure are prepared (provider-specific).
- If there are infrastructure-level restrictions on network communication, ensure that the requirements described in the Network interaction of the platform components section are met.
- There is access to the Deckhouse container registry (official `registry.deckhouse.io`, or a mirror).

Preparing the Configuration

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

Installation configuration file

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

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

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

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

2. ClusterConfiguration: General cluster parameters, such as Kubernetes (control plane components) version, network settings, CRI parameters, etc. **Required**, except when DKP is installed into an already existing Kubernetes cluster.

3. StaticClusterConfiguration: Parameters for a cluster deployed on bare-metal servers (including hybrid clusters) or in unsupported clouds. **Required**, except when DKP is installed into an already existing Kubernetes cluster.

   To add worker node groups (the NodeGroup object), you may also need StaticInstance and SSHCredentials.

4. <PROVIDER>ClusterConfiguration: Parameters for integration with a cloud provider. **Required** when integrating DKP with a supported cloud infrastructure.

   Examples of resources configuring integration with a cloud provider:

   - AWSClusterConfiguration: Amazon Web Services
   - AzureClusterConfiguration: Microsoft Azure
   - DVPClusterConfiguration: Deckhouse Virtualization Platform
   - GCPClusterConfiguration: Google Cloud Platform
   - HuaweiCloudClusterConfiguration: Huawei Cloud
   - OpenStackClusterConfiguration: OpenStack, OVHcloud, Selectel, VK Cloud
   - VsphereClusterConfiguration: VMware vSphere
   - VCDClusterConfiguration: VMware Cloud Director
   - YandexClusterConfiguration: Yandex Cloud
   - ZvirtClusterConfiguration: zVirt

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

5. DKP module configurations.

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

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

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

   Modules often configured during installation:

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

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

6. IngressNginxController: Parameters of the HTTP/HTTPS load balancer (Ingress controller).

7. NodeGroup: Node group parameters. Required to add worker nodes.

8. Objects for authentication and authorization such as ClusterAuthorizationRule, AuthorizationRule, User, Group, and DexProvider.

   See authentication and authorization documentation for details.

An example of the installation config...

- Configuration applicable since DKP 1.75
- Legacy configuration

In this example, access to the DKP container registry is configured using ModuleConfig `deckhouse`.

```
apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
clusterType: Cloud
cloud:
  provider: Azure
  prefix: cloud-demo
podSubnetCIDR: 10.111.0.0/16
serviceSubnetCIDR: 10.222.0.0/16
kubernetesVersion: "Automatic"
clusterDomain: cluster.local
---
```

```
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: <SSH_PUBLIC_KEY>
vNetCIDR: 10.241.0.0/16
subnetCIDR: 10.241.0.0/24
masterNodeGroup:
  replicas: 3
  instanceClass:
    machineSize: Standard_D4ds_v4
    urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140
    enableExternalIP: true
provider:
  subscriptionId: <SUBSCRIPTION_ID>
  clientId: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
  tenantId: <TENANT_ID>
  location: westeurope
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  enabled: true
  settings:
    releaseChannel: Stable
    bundle: Default
    logLevel: Info
    registry:
      mode: Unmanaged
      unmanaged:
        imagesRepo: test-registry.io/some/path
        scheme: HTTPS
        username: <username>
        password: <password>
        ca: <CA>
  version: 1
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  settings:
    modules:
      publicDomainTemplate: "%s.k8s.example.com"
  version: 2
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    controlPlaneConfigurator:
      dexCAMode: DoNotNeed
    publishAPI:
      enabled: true
      https:
        mode: Global
        global:
          kubeconfigGeneratorMasterCA: ""
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-manager
spec:
  version: 1
  enabled: true
```

```
  settings:
    earlyOomEnabled: false
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  # Specify if you plan to use dedicated nodes for monitoring.
  # settings:
  #   nodeSelector:
  #     node.deckhouse.io/group: monitoring
---
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: "nginx"
  inlet: "LoadBalancer"
  nodeSelector:
    node.deckhouse.io/group: worker
---
apiVersion: deckhouse.io/v1
kind: AzureInstanceClass
metadata:
  name: worker
spec:
  machineSize: Standard_F4
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cloudInstances:
    classReference:
      kind: AzureInstanceClass
      name: worker
    maxPerZone: 3
    minPerZone: 1
    zones: ["1"]
  nodeType: CloudEphemeral
---
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: admin
spec:
  subjects:
  - kind: User
    name: admin@deckhouse.io
  accessLevel: SuperAdmin
  portForwarding: true
---
apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  email: admin@deckhouse.io
  password: '$2a$10$isZrV6uzS6F7eGfaNB1EteLTWky7qxJZfbogRs1egWEPuT1XaOGg2'
```

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

```
apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
clusterType: Cloud
cloud:
  provider: Azure
  prefix: cloud-demo
```

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

```
spec:
  version: 1
  enabled: true
  settings:
    earlyOomEnabled: false
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  # Specify if you plan to use dedicated nodes for monitoring.
  # settings:
  #   nodeSelector:
  #     node.deckhouse.io/group: monitoring
---
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: "nginx"
  inlet: "LoadBalancer"
  nodeSelector:
    node.deckhouse.io/group: worker
---
apiVersion: deckhouse.io/v1
kind: AzureInstanceClass
metadata:
  name: worker
spec:
  machineSize: Standard_F4
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cloudInstances:
    classReference:
      kind: AzureInstanceClass
      name: worker
    maxPerZone: 3
    minPerZone: 1
    zones: ["1"]
  nodeType: CloudEphemeral
---
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: admin
spec:
  subjects:
  - kind: User
    name: admin@deckhouse.io
  accessLevel: SuperAdmin
  portForwarding: true
---
apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  email: admin@deckhouse.io
  password: '$2a$10$isZrV6uzS6F7eGfaNB1EteLTWky7qxJZfbogRs1egWEPuT1XaOGg2'
```

Post-bootstrap script

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

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

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

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

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

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

set -e
set -o pipefail


INGRESS_NAME="nginx"


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

# Declare the variable.
lb_ip=""

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

  lb_ip=""

  sleep 5
done

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

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

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

Installing

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

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

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

Where:

1. `<DECKHOUSE_REVISION>`: DKP edition. For example, `ee` for Enterprise Edition, `ce` for Community Edition, etc.
2. `<MOUNT_OPTIONS>`: Parameters for mounting files into the installer container, such as:
   - SSH access keys
   - Configuration file
   - Resource file, etc.
3. `<RELEASE_CHANNEL>`: Release channel in kebab-case format:
   - `alpha`: Alpha channel

- `beta`: Beta channel
- `early-access`: Early Access channel
- `stable`: Stable channel
- `rock-solid`: Rock Solid channel

Here is an example of a command to run the DKP Community Edition installer container from the Stable release channel:

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

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

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

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

Example of running a DKP installation with cloud cluster deployment:

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

Where:

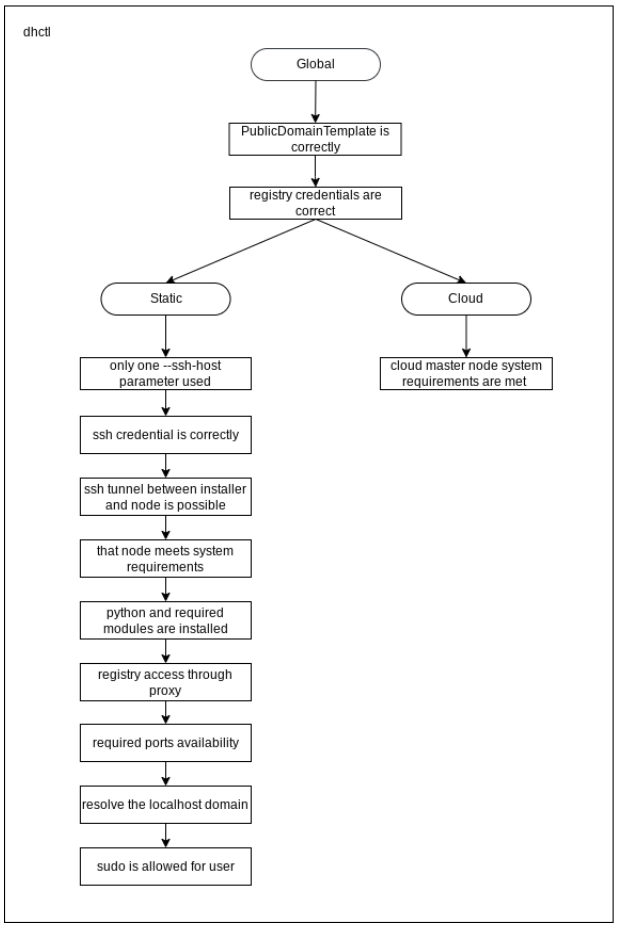
- `/config.yml`: Installation configuration file.
- `<SSH_USER>`: Username for SSH connection to the server.
- `--ssh-agent-private-keys`: Private SSH key file for SSH connection.
- `<SSH_PRIVATE_KEY_FILE>`: Name of private key. For example, for a key with RSA encryption it can be `id_rsa`, and for a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

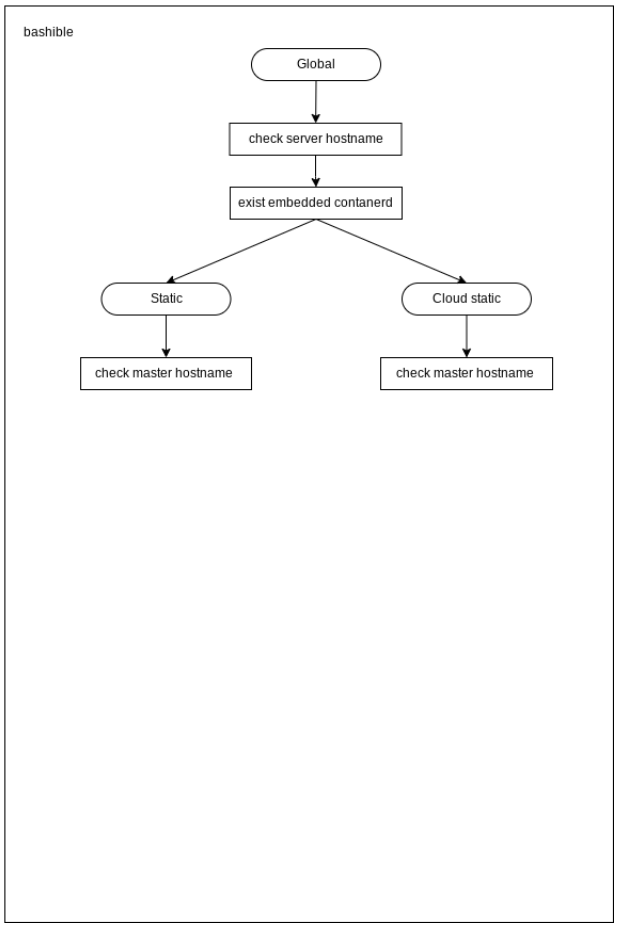
Pre-installation checks

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

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

Preflight checks





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

1. General checks:
   - The values of the parameters `publicDomainTemplate` and `clusterDomain` do not match.
   - The authentication data for the container registry specified in the installation configuration is correct.
   - The host name meets the following requirements:
     - The length does not exceed 63 characters.
     - It consists only of lowercase letters.
     - It does not contain special characters (hyphens `-` and periods `.` are allowed, but they cannot be at the beginning or end of the name).
   - The server (VM) has a supported container runtime (`containerd`) installed.
   - The host name is unique within the cluster.
   - The server's system time is correct.
   - The address spaces for Pods (`podSubnetCIDR`) and services (`serviceSubnetCIDR`) do not intersect.
2. Checks for static and hybrid cluster installation:
   - Only one `--ssh-host` parameter is specified. For static cluster configuration, only one IP address can be provided for configuring the first master node.
   - SSH connection is possible using the specified authentication data.
   - SSH tunneling to the master node server (or VM) is possible.
   - The server (VM) selected for the master node installation must meet the minimum system requirements:
     - At least 4 CPU cores.
     - At least 8 GB of RAM.
     - At least 60 GB of disk space with 400+ IOPS performance.
     - Linux kernel version 5.8 or newer.
     - One of the package managers installed: `apt`, `apt-get`, `yum`, or `rpm`.
     - Access to standard OS package repositories.
     - **When using `ContainerdV2`** as the default container runtime on cluster nodes:
       - Support for `CgroupsV2`.
       - Systemd version `244`.
       - Support for the `erofs` kernel module.
   - Python (Python 3 or Python 2) must be available on the server (VM) intended for the master node. The standard Python modules used by the installer must also be available:
     - For HTTP request handling: `urllib.request` (Python 3) or `urllib2` (Python 2)
     - For HTTP error handling: `urllib.error` (Python 3) or `urllib2` (Python 2)
     - For working with configuration files: `configparser` (Python 3) or `ConfigParser` (Python 2)
     - For HTTP server: `http.server` (Python 3) or `SimpleHTTPServer` (Python 2)
     - For TCP server: `http.server` (Python 3) or `SocketServer` (Python 2)
   - The container registry is accessible through a proxy (if proxy settings are specified in the installation configuration).
   - On the server (VM) intended for the master node and on the host from which the installer is launched, port `22/TCP` (network access from a personal computer) must be accessible, as it is required for the installation process. For details, see the Network interaction of the platform components section.
   - DNS must resolve `localhost` to IP address `127.0.0.1`.

    - The user has `sudo` privileges on the server (VM).
    - Required ports for the installation must be open:
        - Port `22/TCP` between the host running the installer and the server.
        - No port conflicts with those used by the installation process.
    - The server (VM) has the correct time.
    - The user `deckhouse` must not exist on the server (VM).
    - The address spaces for Pods (`podSubnetCIDR`), services (`serviceSubnetCIDR`), and internal network (`internalNetworkCIDRs`) do not intersect.
3. Checks for cloud cluster installation:
    - The configuration of the virtual machine for the master node meets the minimum requirements.
    - The cloud provider API is accessible from the cluster nodes.
    - The configuration for Yandex Cloud with NAT Instance is verified.

List of preflight skip flags...

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

- `--preflight-skip-all-checks`: Skip all preflight checks.
- `--preflight-skip-check=static-ssh-tunnel`: Skip the SSH forwarding check.
- `--preflight-skip-check=ports-availability`: Skip the check for the availability of required ports.
- `--preflight-skip-check=resolve-localhost`: Skip the `localhost` resolution check.
- `--preflight-skip-check=dhctl-edition`: Skip the DKP version check.
- `--preflight-skip-check=registry-access-through-proxy`: Skip the check for accessing the registry through a proxy server.
- `--preflight-skip-check=public-domain-template`: Skip the check for the `publicDomain` template.
- `--preflight-skip-check=static-ssh-credential`: Skip the check for SSH user credentials.
- `--preflight-skip-check=registry-credentials`: Skip the check for registry access credentials.
- `--preflight-skip-check=python-modules`: Skip the check for Python installation.
- `--preflight-skip-check=sudo-allowed`: Skip the check for `sudo` privileges.
- `--preflight-skip-check=static-system-requirements`: Skip the check for meeting system requirements.
- `--preflight-skip-check=static-single-ssh-host`: Skip the check for the number of specified SSH hosts.
- `--preflight-skip-check=cloud-api-accessibility`: Skip the Cloud API accessibility check.
- `--preflight-skip-check=time-drift`: Skip the time drift check.
- `--preflight-skip-check=cidr-intersection`: Skip the CIDR intersection check.
- `--preflight-skip-check=deckhouse-user`: Skip the `deckhouse` user existence check.
- `--preflight-skip-check=yandex-cloud-config`: Skip the Yandex Cloud with NAT Instance configuration check.
- `--preflight-skip-check=dvp-kubeconfig`: Skip the DVP kubeconfig check.
- `--preflight-skip-check=static-instances-ssh-credentials` — skip verifying accessibility StaticInstances with SSHCredentials.

Example of using a preflight skip flag:

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

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

Aborting the installation

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

```
dhctl bootstrap-phase abort
```

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

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

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

Installing from an external (third-party) registry

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

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

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

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

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

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

Example:

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

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

Set the following parameters in `InitConfiguration`:

- `imagesRepo: <PROXY REGISTRY>/<DECKHOUSE REPO PATH>/ee`: Path to the DKP EE image in an external registry. Example: `imagesRepo: registry.deckhouse.io/deckhouse/ee`.
- `registryDockerCfg: <BASE64>`: Base64-encoded access credentials to the external registry.

If anonymous access is allowed to DKP images in the external registry, the `registryDockerCfg` should look as follows:

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

The provided value must be Base64-encoded.

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

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

Where:

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

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

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

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

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

echo "$MYRESULTSTRING"
```

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

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

Nexus configuration notes

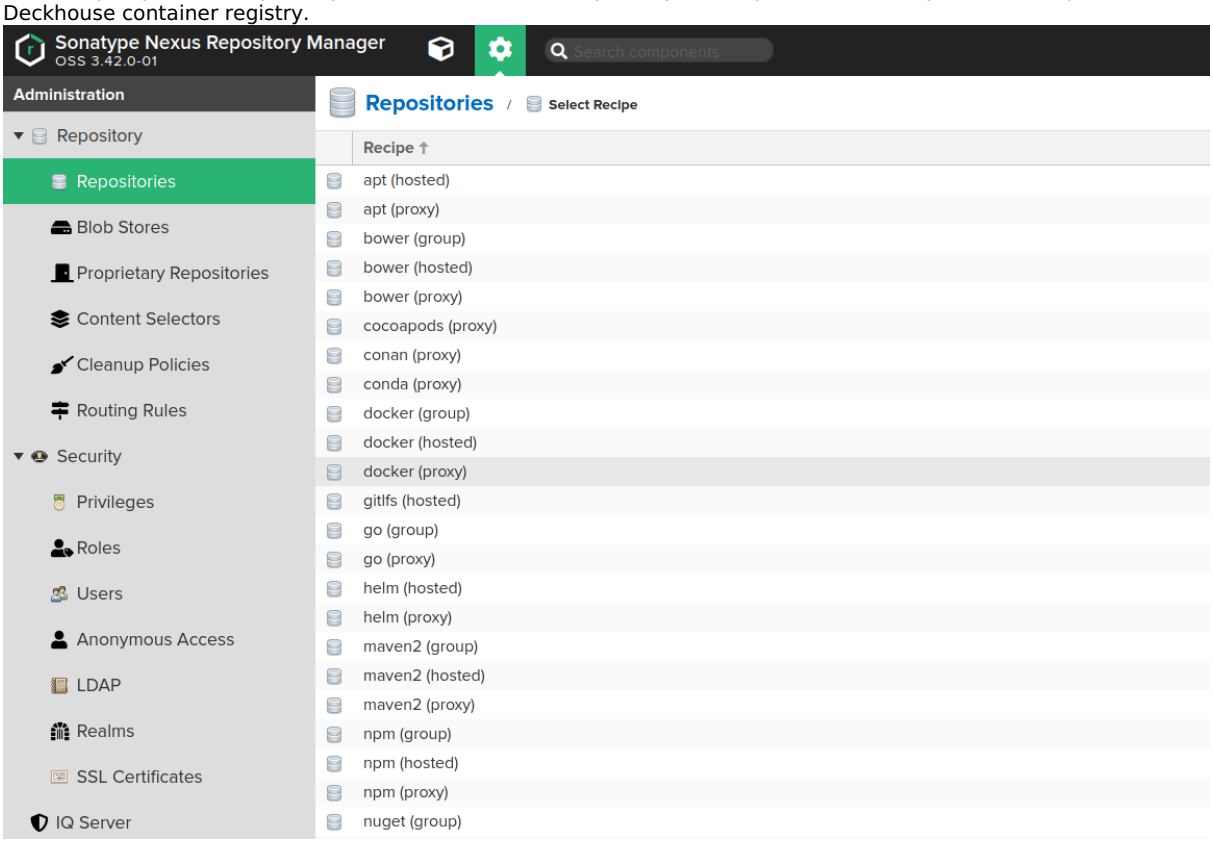
> ⚠ When interacting with a `docker`-type repository in Nexus (for example, using `docker pull` or `docker push` commands), you must specify the address in the following format: `<NEXUS_URL>:<REPOSITORY_PORT>/<PATH>`.
>
> Using the `URL` value from the Nexus repository settings is **not allowed**.

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

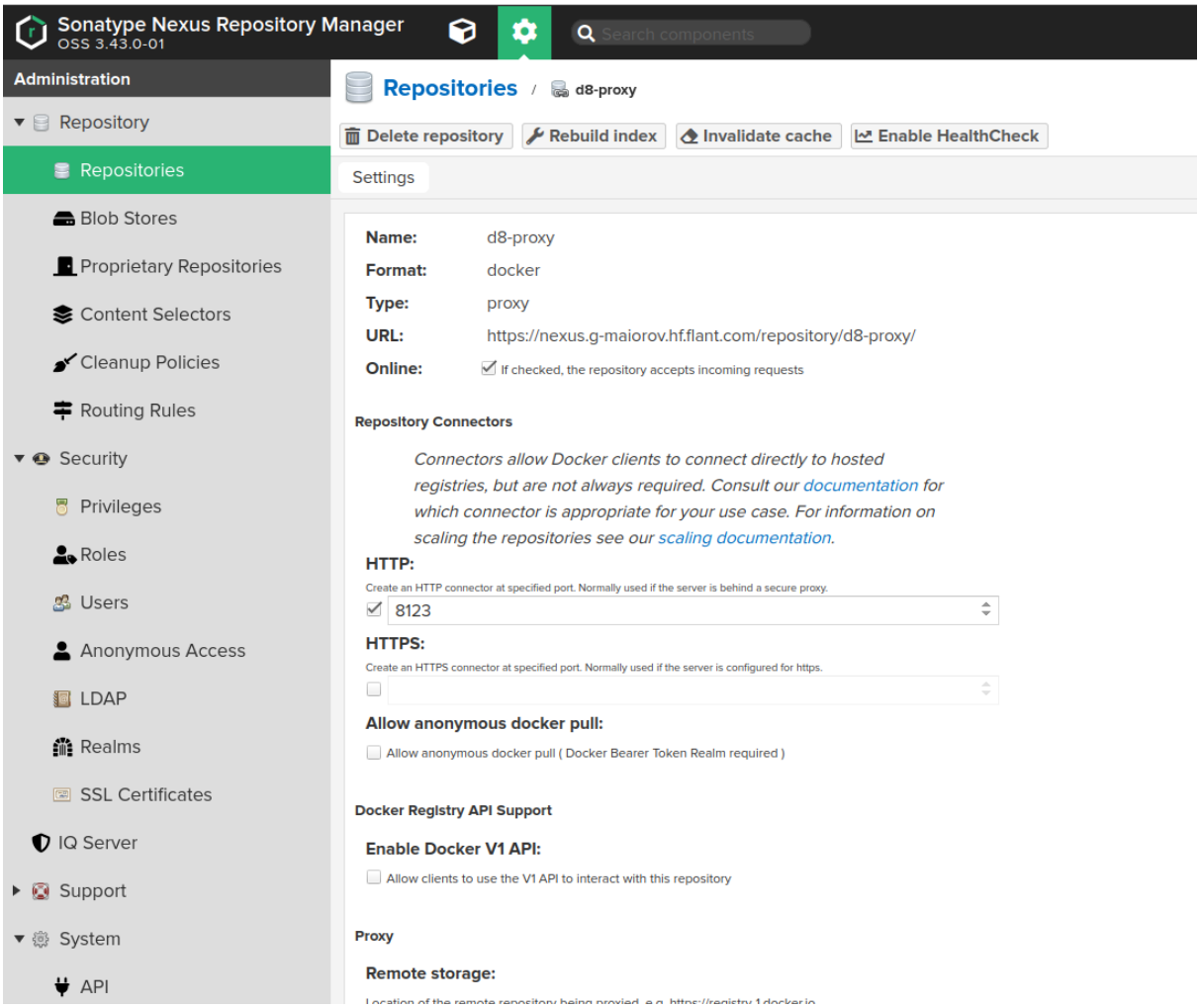
- A proxy Docker repository must be created ("Administration" → "Repository" → "Repositories"):
  - The `Maximum metadata age` parameter is set to `0`.
- Access control must be configured:
  - Create a role named **Nexus** ("Administration" → "Security" → "Roles") with the following privileges:
    -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

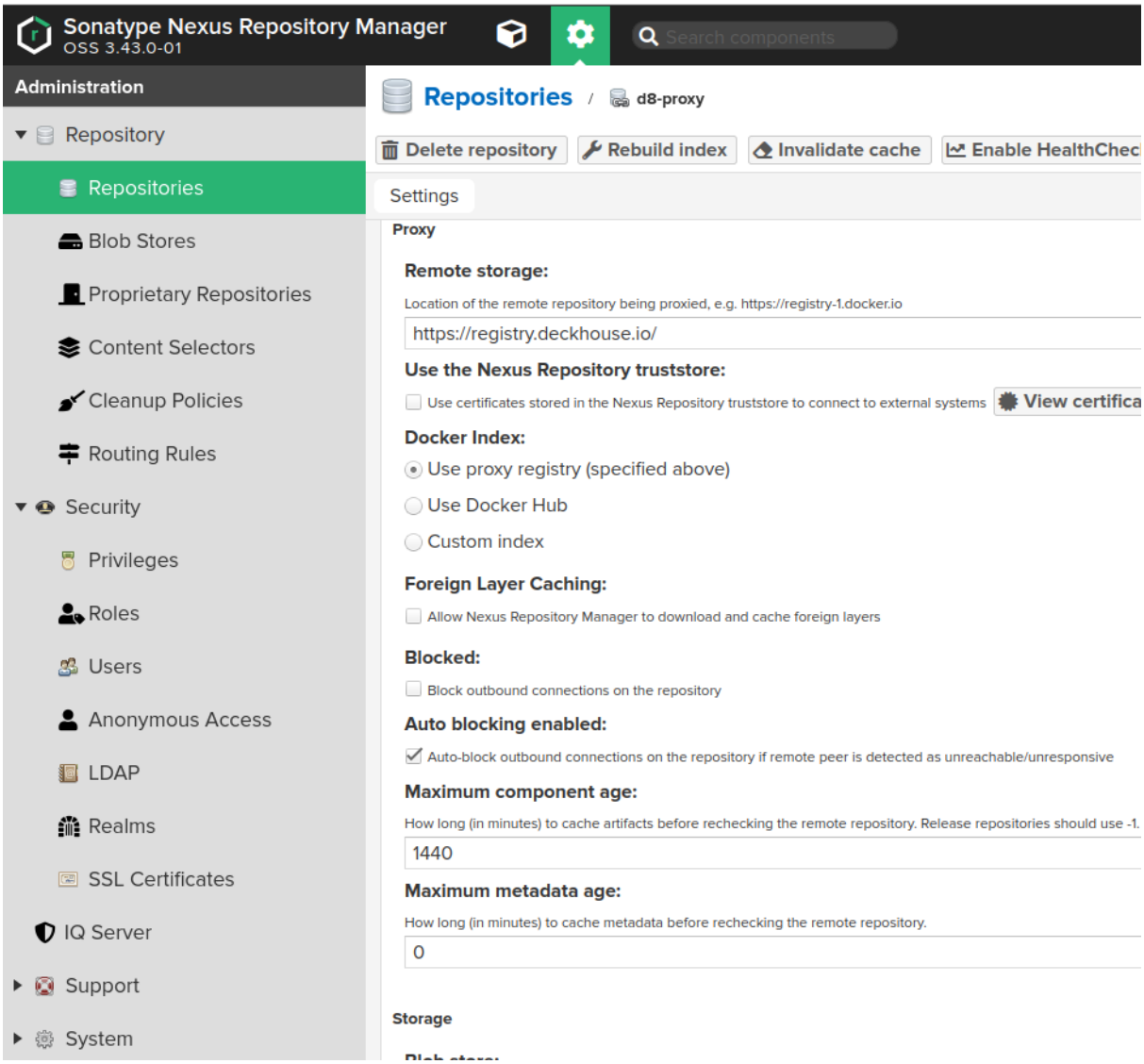
To configure Nexus, follow these steps:

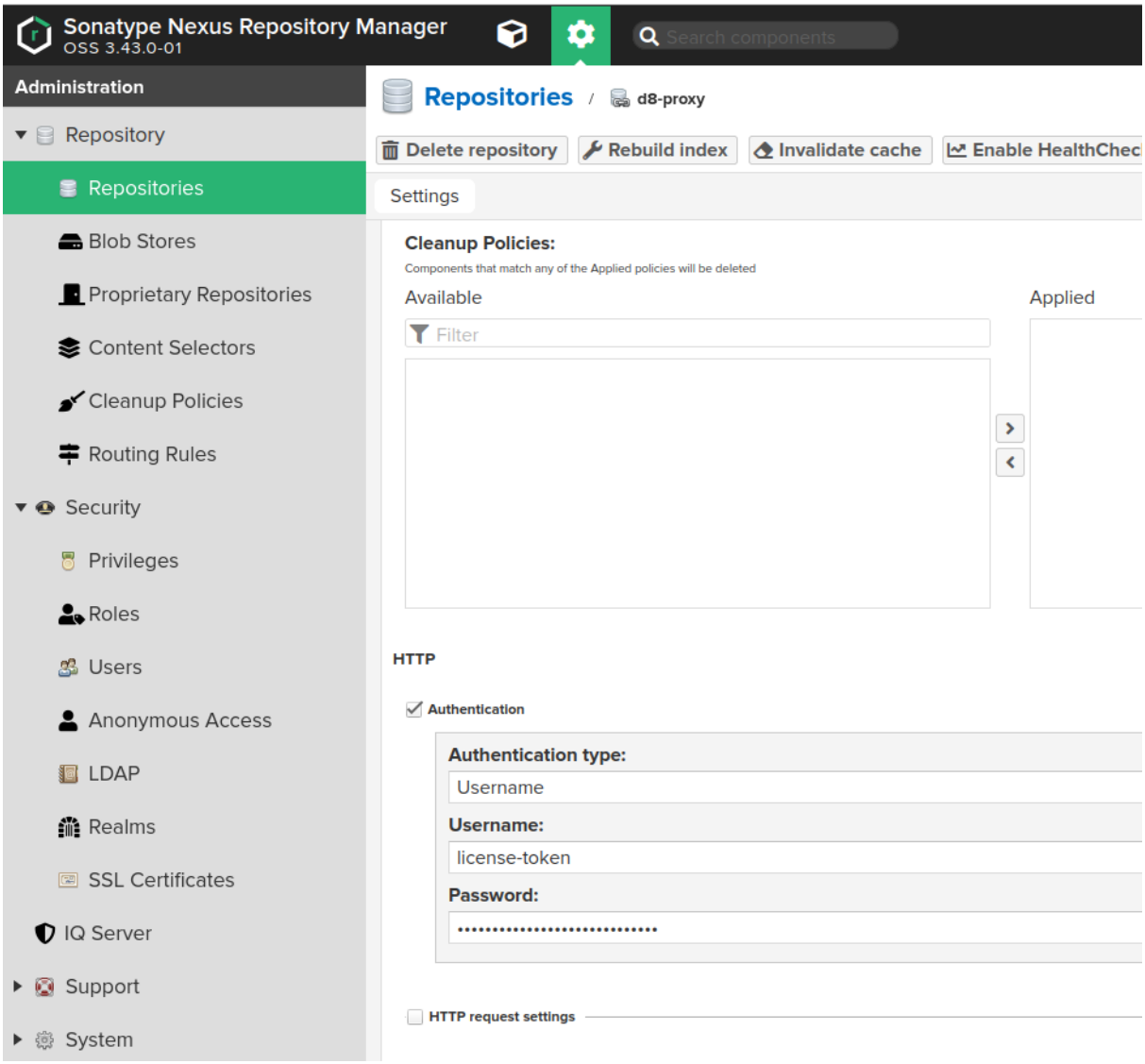
1. Create a proxy Docker repository ("Administration" → "Repository" → "Repositories") that points to the public Deckhouse container registry.



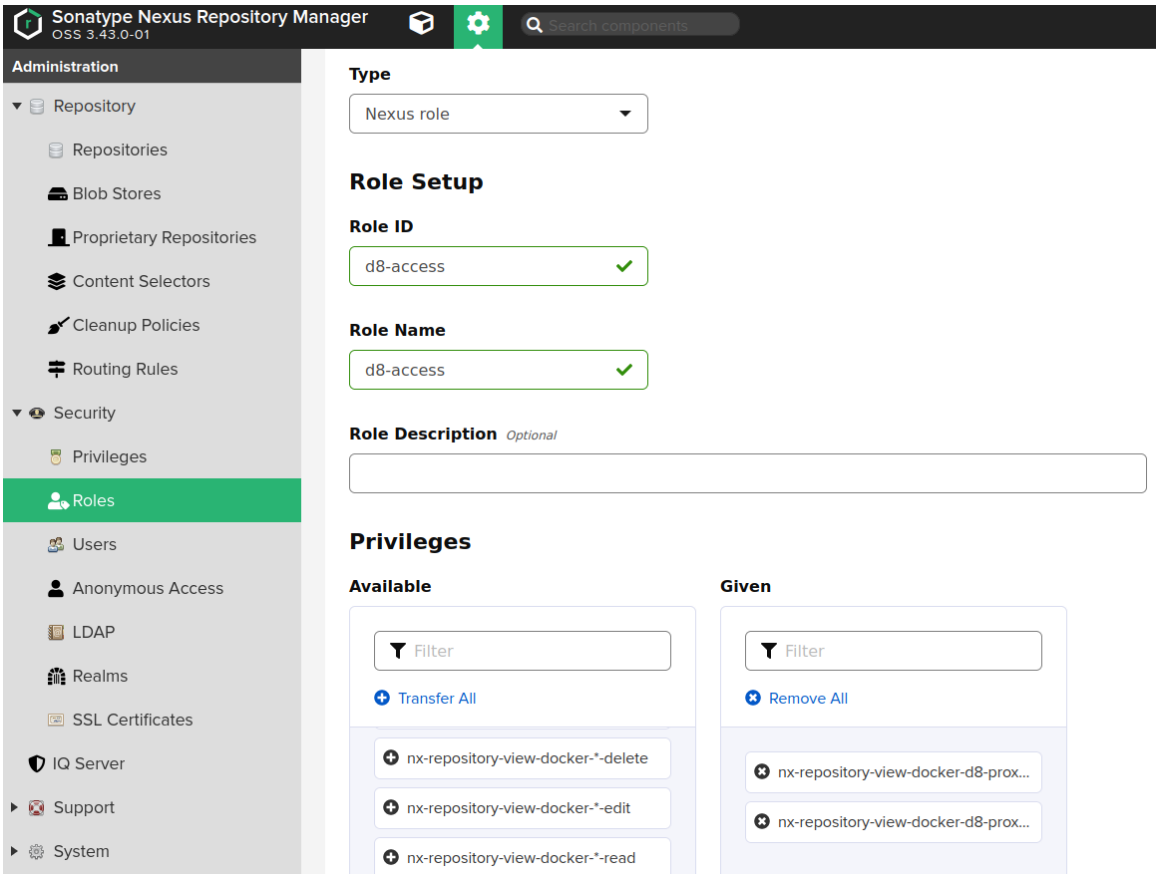
2. Fill out the repository creation form with the following values:
   - `Name`: the desired repository name, e.g., `d8-proxy`.
   - `Repository Connectors / HTTP` or `HTTPS`: a dedicated port for the new repository, e.g., `8123` or another.
   - `Remote storage`: must be set to `https://registry.deckhouse.io/`.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


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

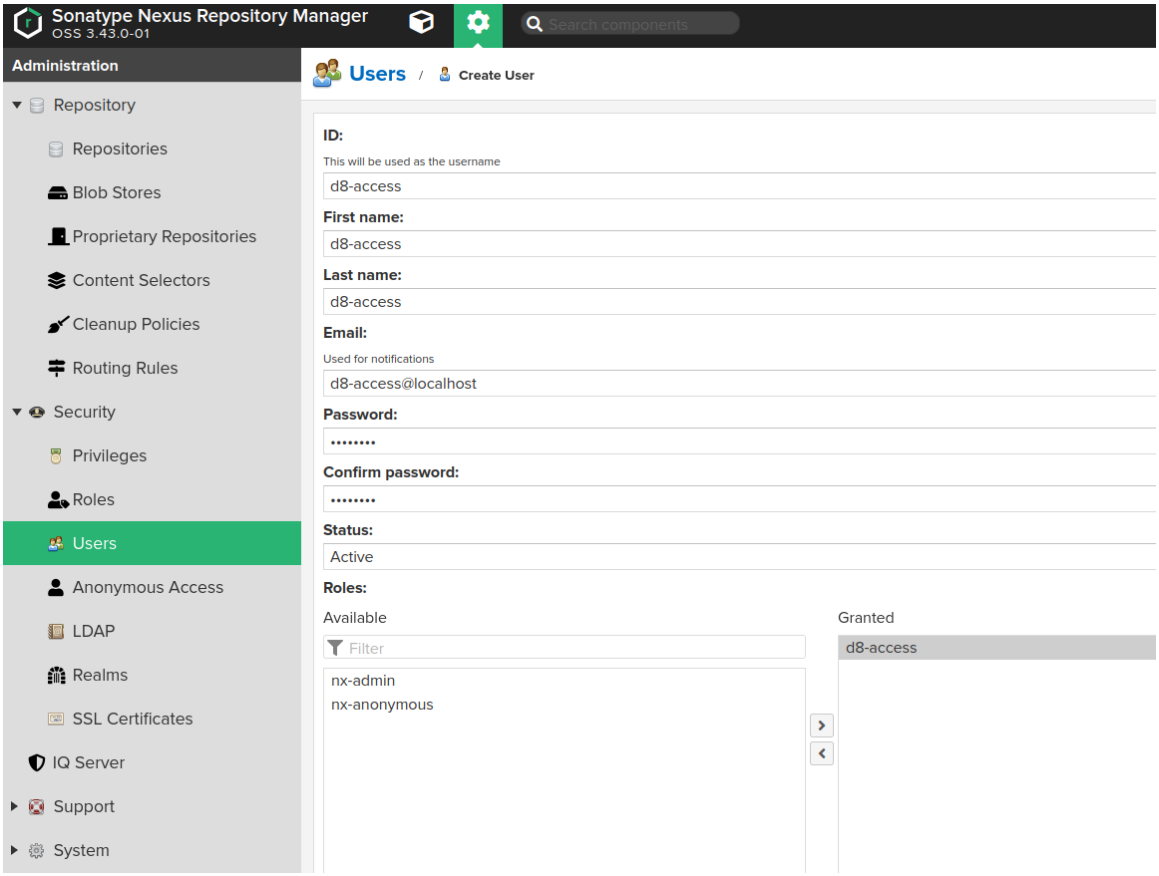

Sonatype Nexus Repository Manager
OSS 3.43.0-01
Search components
Administration
Repository
Repositories
Blob Stores
Proprietary Repositories
Content Selectors
Cleanup Policies
Routing Rules
Security
Privileges
Roles
Users
Anonymous Access
LDAP
Realms
SSL Certificates
IQ Server
Support
System
Repositories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



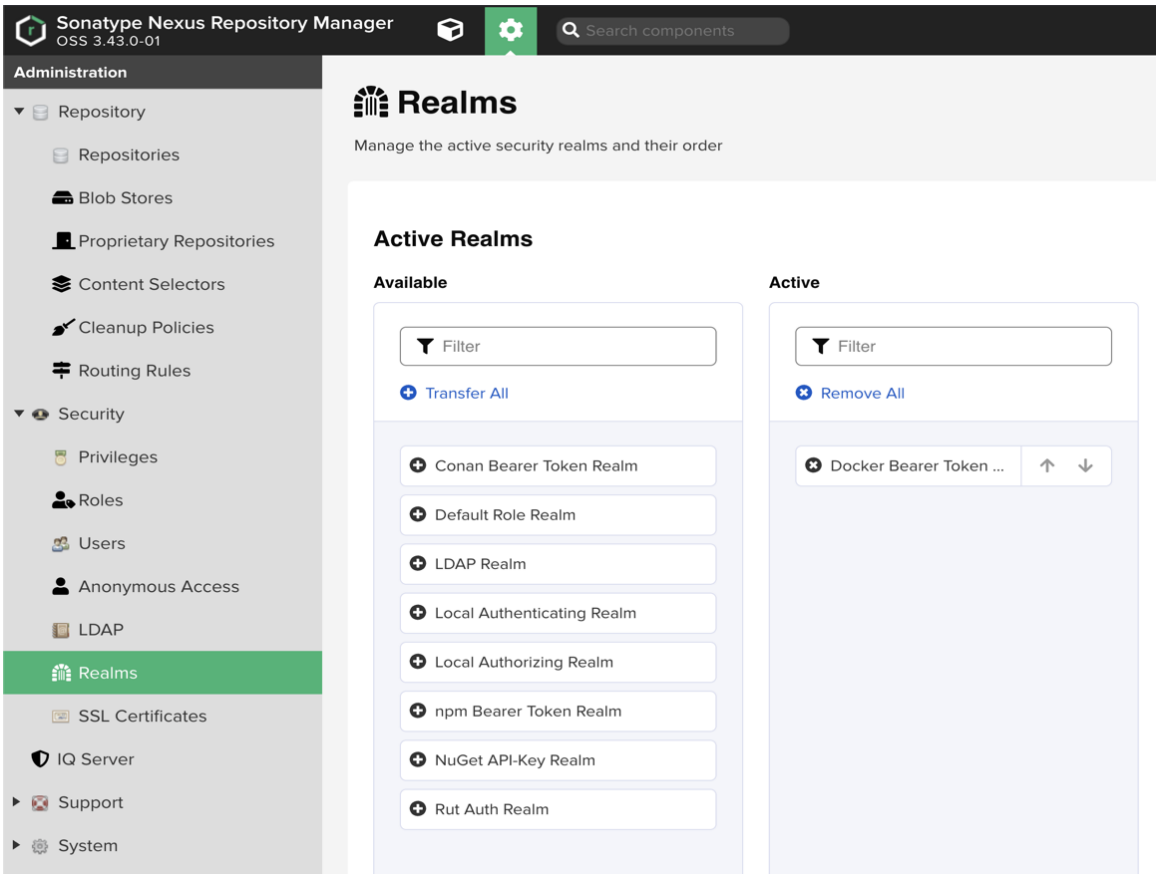
3. Configure Nexus access control to allow DKP to access the created repository:
   - Create a **Nexus** role (“Administration” → “Security” → “Roles”) with the following privileges: `nx-repository-view-docker-<repository>-browse` and `nx-repository-view-docker-<repository>-read`.



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



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

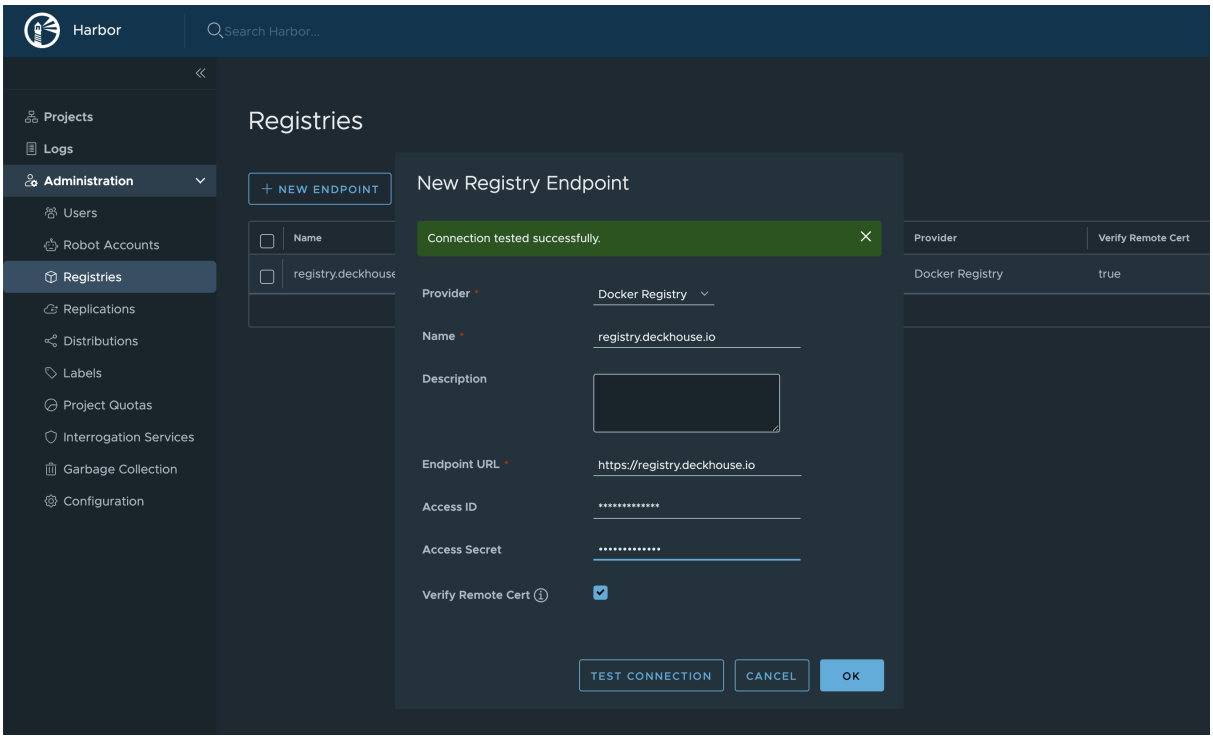


As a result, DKP images will be available at a URL as follows: `https://<NEXUS_HOST>:<REPOSITORY_PORT>/deckhouse/ee:<d8s-version>`.

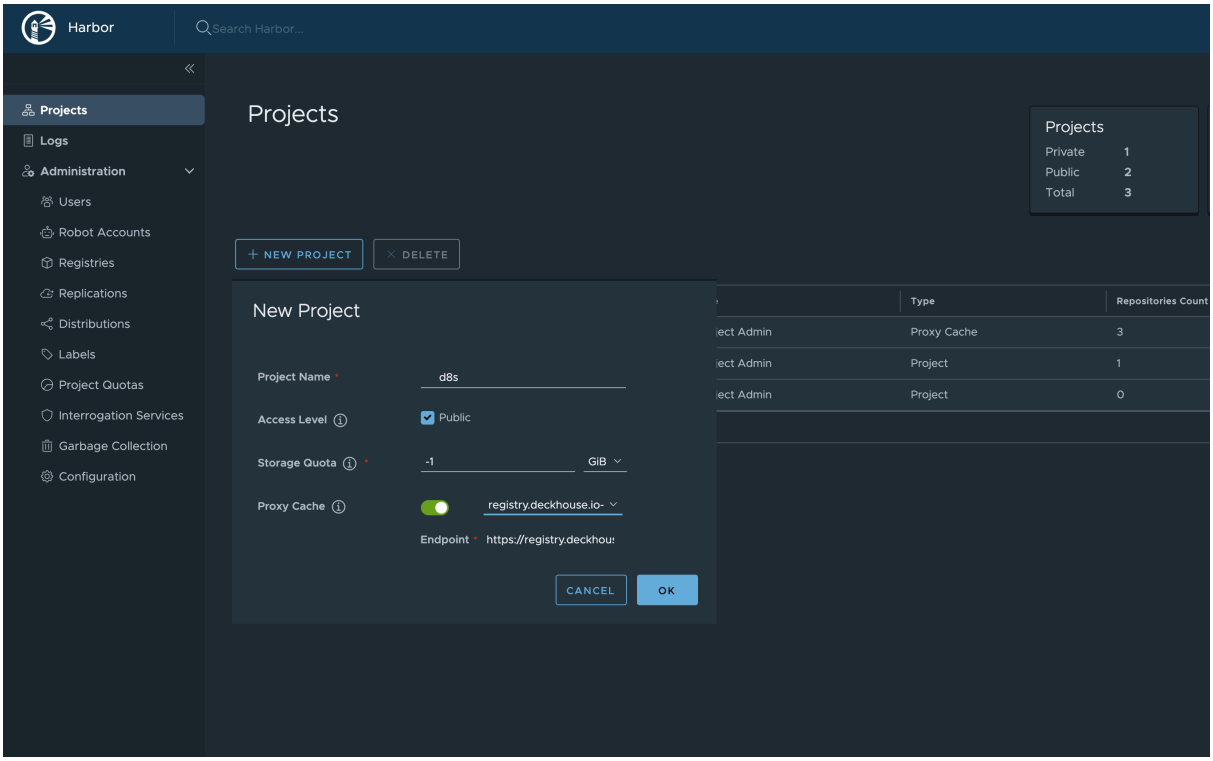
Harbor configuration notes

Use the Harbor Proxy Cache feature.

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



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



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

Manual loading of DKP images and vulnerability DB into a private registry

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

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

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

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

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

  Where:

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

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

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

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

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

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

Example command to download DKP images from an external registry:

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

Example command to download the vulnerability scanner database package:

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

Example command to download all available additional module packages:

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

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

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

Example command to download `stronghold` module with semver `^` constraint from version 1.2.0:

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

Example command to download `secrets-store-integration` module with semver `~` constraint from version 1.1.0:

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

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

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

Other command parameters available for use:

- `--no-pull-resume`: Force the download to start from the beginning.
- `--no-platform`: Skip downloading the Deckhouse Kubernetes Platform image package (`platform.tar`).
- `--no-modules`: Skip downloading module packages (`module-*.tar`).
- `--no-security-db`: Skip downloading the vulnerability scanner database package (`security.tar`).
- `--include-module` / `-i` = `name[@Major.Minor]`: Download only a specific set of modules using a whitelist (and, if needed, their minimum versions). Use multiple times to add more modules to the whitelist. These flags are ignored if used with `--no-modules`.

  The following syntax options are supported for specifying module versions:

    - `module-name@1.3.0`: Pulls versions with semver ^ constraint (^1.3.0), including v1.3.0, v1.3.3, v1.4.1.
    - `module-name@~1.3.0`: Pulls versions with semver ~ constraint (>=1.3.0 <1.4.0), including only v1.3.0, v1.3.3.
    - `module-name@=v1.3.0`: Pulls exact tag match v1.3.0, publishing to all release channels.
    - `module-name@=bobV1`: Pulls exact tag match “bobV1”, publishing to all release channels.
- `--exclude-module` / `-e =name`: Skip downloading a specific set of modules using a blacklist. Use multiple times to add more modules to the blacklist. Ignored if `--no-modules` or `--include-module` is used;
- `--modules-path-suffix`: Change the suffix of the path to the module repository in the main DKP registry. The default suffix is `/modules` (for example, the full path to the module repo will be `registry.deckhouse.io/deckhouse/EDITION/modules`).
- `--since-version=X.Y`: Download all DKP versions starting from the specified minor version. This option is ignored if the specified version is higher than the version on the Rock Solid release channel. Cannot be used with `--deckhouse-tag`.
- `--deckhouse-tag`: Download only the specific DKP version (regardless of release channels). Cannot be used with `--since-version`.
- `--gost-digest`: Calculate the checksum of the final DKP image bundle using the GOST R 34.11-2012 (Streebog) algorithm. The checksum will be displayed and written to a `.tar.gostsum` file in the folder containing the image tarball.
- `--source-login` and `--source-password`: Authentication data to access the external registry.
- `--images-bundle-chunk-size=N`: The maximum file size (in GB) to split the image archive into. As a result, instead of one image archive, a set of CHUNK files will be created (for example, `d8.tar.NNNN.chunk`). To upload images from such a set, use the file name without the `.NNNN.chunk` suffix (for example, `d8.tar` for files `d8.tar.NNNN.chunk`).
- `--tmp-dir`: Path to a directory for temporary files used during image download and upload. All processing is done in this directory. It must have enough free disk space to hold the entire image bundle. Defaults to the `.tmp` subdirectory in the image bundle directory.

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

- `HTTP_PROXY` / `HTTPS_PROXY`: Proxy server URL for HTTP(S) requests not listed in the `$NO_PROXY` variable.
- `NO_PROXY`: Comma-separated list of hosts to exclude from proxying. Each entry can be an IP (`1.2.3.4`), CIDR (`1.2.3.4/8`), domain, or wildcard (`*`). IPs and domains may include a port (`1.2.3.4:80`). A domain matches itself and all subdomains. A domain starting with a `.` matches only subdomains. For example, `foo.com` matches `foo.com` and `bar.foo.com`; `.y.com` matches `x.y.com` but not `y.com`. The `*` disables proxying.
- `SSL_CERT_FILE`: Path to an SSL certificate. If set, system certificates are not used.
- `SSL_CERT_DIR`: Colon-separated list of directories to search for SSL certificate files. If set, system certificates are not used. More info...
- `MIRROR_BYPASS_ACCESS_CHECKS`: Set this variable to `1` to disable credential validation for the registry.

- On the host with access to the container registry where DKP images should be uploaded, copy the downloaded DKP image bundle and install the Deckhouse CLI.

- Upload DKP images to the registry using the `d8 mirror push` command.

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

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

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

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

  If you’re using Harbor, you won’t be able to upload images to the root of a project. Use a dedicated repository within the project to store DKP images.

- After uploading the images to the registry, you can proceed with installing DKP. Use the getting-started guide.

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

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

Creating a cluster and running DKP without using release channels

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

If you need to install DKP with automatic updates disabled:

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

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

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

Using a proxy server

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

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

   ```
   apt-get install squid
   ```

3. Create the Squid configuration file:

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

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

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

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

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

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

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

Example:

```
apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
clusterType: Cloud
cloud:
  provider: OpenStack
  prefix: main
podSubnetCIDR: 10.111.0.0/16
serviceSubnetCIDR: 10.222.0.0/16
kubernetesVersion: "Automatic"
cri: "Containerd"
clusterDomain: "cluster.local"
proxy:
  httpProxy: "http://user:password@proxy.company.my:3128"
  httpsProxy: "https://user:password@proxy.company.my:8443"
```

Automatic proxy variable loading for users in CLI

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

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

Additional resources

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

How to configure?

Deckhouse configuration

You can configure Deckhouse using:

- **Global settings**. Global settings are stored in the `ModuleConfig/global` custom resource. Global settings can be be thought of as a special `global` module that cannot be disabled.
- **Module settings**. Module settings are stored in the `ModuleConfig` custom resource; its name is the same as that of the module (in kebab-case).
- **Custom resources.** Some modules are configured using the additional custom resources.

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

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

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

```
d8 k get moduleconfigs
```

Example output...

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

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

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

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

Changes are applied automatically once the resource configuration is saved.

Modifying cluster configuration

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

General cluster parameters are defined in the ClusterConfiguration structure.

To modify these parameters, run the following command:

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

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

Modifying protected parameters

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

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

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

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

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

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

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

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

Modifying protected parameters using dhctl

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

It will automatically:

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

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

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

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

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

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

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

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

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

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

   Where:

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

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

Manually changing protected parameters

Manually edit the configuration:

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

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

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

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

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

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

4. Encode the edited configuration and update the Secret:

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

5. Remove the annotation after applying the changes:

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

6. Delete the temporary file:

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

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

Viewing current configuration

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

1. To view global settings, run:

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

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

```
d8 k get modules
```

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

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

Configuring the module

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

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

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

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

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

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

Enabling and disabling the module

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

To enable/disable the module, set `spec.enabled` field of the ModuleConfig custom resource to `true` or `false`. Note that this may require you to first create a ModuleConfig resource for the module.

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

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

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

Example:

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

Module bundles

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

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

Bundle name	List of modules, enabled by default

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

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

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

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

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

- cloud provider module (for example, `cloud-provider-aws` for AWS), in a case of deploying a cloud cluster;
- `cni-cilium` or another CNI control module (if necessary);
- `control-plane-manager`;

- `kube-dns`;
- `node-manager`;
- `registry-packages-proxy`;
- `terraform-manager`, in a case of deploying a cloud cluster.

Accessing documentation for the current version

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

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

Managing placement of Deckhouse components

Advanced scheduling

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

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

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

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

Module features that depend on its type

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

- The *monitoring*-related modules (`operator-prometheus`, `prometheus` and `vertical-pod-autoscaler`):
  - Deckhouse examines nodes to determine a `nodeSelector` in the following order:
    1. It checks if a node with the `node-role.deckhouse.io/MODULE NAME` label is present in the cluster.
    2. It checks if a node with the `node-role.deckhouse.io/monitoring` label is present in the cluster.
    3. It checks if a node with the `node-role.deckhouse.io/system` label is present in the cluster.
  - Tolerations to add (note that tolerations are added all at once):
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"MODULE NAME"}` (e.g., `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"operator-prometheus"}`).
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"monitoring"}`.
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"system"}`.
- The *frontend*-related modules (`ingress-nginx` only):
  - Deckhouse examines nodes to determine a `nodeSelector` in the following order:
    1. It checks if a node with the `node-role.deckhouse.io/MODULE NAME` label is present in the cluster.
    2. It checks if a node with the `node-role.deckhouse.io/frontend` label is present in the cluster.
  - Tolerations to add (note that tolerations are added all at once):
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"MODULE NAME"}`.
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"frontend"}`.
- Other modules:
  - Deckhouse examines nodes to determine a `nodeSelector` in the following order:
    1. It checks if a node with the `node-role.deckhouse.io/MODULE_NAME` label is present in the cluster (e.g., `node-role.deckhouse.io/cert-manager`).
    2. It checks if a node with the `node-role.deckhouse.io/system` label is present in the cluster.
  - Tolerations to add (note that tolerations are added all at once):
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"MODULE NAME"}` (e.g., `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"network-gateway"}`).
    - `{"key":"dedicated.deckhouse.io","operator":"Equal","value":"system"}`.

Additional resources

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

Working with the image registry

This section describes how to work with image registries and editions in Deckhouse Kubernetes Platform.

Working with image registries

This section describes working with image registries in a running cluster. If you are interested in information about working with the registry when installing the cluster, go to the Installing the Platform section.

The section Third-party registry describes the process of switching a running DKP cluster to use a third-party registry. The section Internal registry describes preparing to switch between cluster operating modes: with or without using the internal container image registry, as well as the switching processes. The section Restoring registry access describes the process for resuming Deckhouse Kubernetes Platform image downloads after a license token has expired is described.

Working with editions

The section Switching editions describes the possible options for switching editions in a running Deckhouse Kubernetes Platform cluster.

Switching a running DKP cluster to use an external registry

⚠ When using the registry module, change the address and parameters of the registry in the registry section of the `deckhouse` module configuration. An example of configuration is provided in the `registry` module documentation.

⚠ If, during the switching process, the image of a module did not reload and the module did not reinstall, use the instructions to resolve the issue.

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

1. Run the `deckhouse-controller helper change-registry` command from the DKP pod with the parameters of the new registry.
   Example:

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

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

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

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

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

   Example output:

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

3. Wait until the registry pod reaches the `Ready` status. If the pod is in the `ImagePullBackoff` state, restart it.
4. Wait for bashible to apply the new settings on the master node.

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

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

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

   Example of output when viewing the bashible service log:

   ```
   $ journalctl -u bashible -n 20
   ...
   Aug 13 05:03:08 kube-master-0 systemd[1]: Started Bashible service.
   Aug 13 05:03:10 kube-master-0 bash[1847265]: Configuration is in sync, nothing to do.   <--
   Aug 13 05:03:10 kube-master-0 systemd[1]: bashible.service: Deactivated successfully.
   Aug 13 05:03:10 kube-master-0 systemd[1]: bashible.service: Consumed 1.075s CPU time.
   ```

5. If you need to disable automatic registry updates via the external registry, remove the `releaseChannel` parameter from the `deckhouse` module configuration.
6. Check if any pods in the cluster are still using the original registry address:

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

Additional resources

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

Managing the internal container image registry

The ability to use internal registry is implemented by the `registry` module.

The internal registry allows for optimizing the downloading and storage of images, as well as helping to ensure availability and fault tolerance for Deckhouse Kubernetes Platform.

Modes of operation with the internal registry

The `registry` module, which implements internal storage, operates in the following modes:

- `Direct`: Enables the internal container image registry. Access to the internal registry is performed via the fixed address `registry.d8-system.svc:5001/system/deckhouse`. This fixed address allows Deckhouse images to avoid being re-downloaded and components to avoid being restarted when registry parameters change. Switching between modes and registries is done through the `deckhouse` ModuleConfig. The switching process is automatic (for more details, see the switching examples below) for more information. The architecture of the mode is described in the section Direct Mode Architecture.
- `Unmanaged`: Operation without using the internal registry. Access within the cluster is performed directly to the external registry. There are two types of the `Unmanaged` mode:
  - Configurable: A mode managed via the `registry` module. Switching between modes and registries is handled through the ModuleConfig of `deckhouse`. The switch is performed automatically (for more details, see the switching examples below).
  - Non-configurable (deprecated): The default mode. Configuration parameters are set during cluster installation or changed in a running cluster using the (deprecated) `helper change registry` command.

- The `Direct` mode requires using the `Containerd` or `Containerd V2` CRI on all cluster nodes. For CRI setup, refer to the `ClusterConfiguration`.

Restrictions on working with the internal registry

Working with the internal registry using the `registry` module has a number of limitations and restrictions concerning installation, operating conditions, and mode switching.

Registry configuration via the `deckhouse` moduleConfig during DKP cluster bootstrap is not supported.

Operating conditions restrictions

The `registry` module works under the following conditions:

- If CRI containerd or containerd v2 is used on the cluster nodes. To configure CRI, refer to the ClusterConfiguration configuration.
- The cluster is fully managed by DKP. The module will not work in Managed Kubernetes clusters.

Mode switching restrictions

Mode switching restrictions are as follows:

- For the first switch, migration of user registry configurations must be performed. For more details, see the Registry Module: FAQ section.
- Switching to the non-configurable `Unmanaged` mode is only available from the `Unmanaged` mode. For more details, see the Registry Module: FAQ section.

Examples of switching

⚠ If, during the switching process, the image of a module did not reload and the module did not reinstall, use the instructions to resolve the issue.

Switching to Direct Mode

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

✖ When changing the registry mode or registry parameters, Deckhouse will be restarted.

1. Before switching, perform the migration to use the `registry` module.
2. Make sure the `registry` module is enabled and running. To do this, execute the following command:

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

   Example output:

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

3. Make sure all master nodes are in the `Ready` state and do not have the `SchedulingDisabled` status, using the following command:

   ```
   d8 k get nodes
   ```

   Example output:

   ```
   NAME       STATUS   ROLES                  ...
   master-0   Ready    control-plane,master   ...
   master-1   Ready    control-plane,master   ...
   master-2   Ready    control-plane,master   ...
   ```

   Example of output when the master node (`master-2` in the example) is in the `SchedulingDisabled` status:

   ```
   NAME       STATUS                     ROLES                 ...
   master-0   Ready    control-plane,master  ...
   master-1   Ready    control-plane,master  ...
   master-2   Ready,SchedulingDisabled    control-plane,master  ...
   ```

4. Ensure the Deckhouse job queue is empty and contains no errors:

   ```
   d8 system queue list
   ```

Example output:

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

5. Set the `Direct` mode configuration in the ModuleConfig `deckhouse`. If you're using a registry other than `registry.deckhouse.io`, refer to the `deckhouse` module documentation for correct configuration.

   Configuration example:

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

6. Check the registry switch status in the `registry-state` secret using this guide.

   Example output:

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

Switching to Unmanaged Mode

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

> ✖ Changing the registry mode or its parameters will cause Deckhouse to restart.

1. Before switching, perform the migration to use the `registry` module.
2. Make sure the `registry` module is enabled and running. To do this, execute the following command:

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

   Example output:

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

3. Ensure the Deckhouse job queue is empty and contains no errors:

```
d8 system queue list
```

   Example output:

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

4. Set the `Unmanaged` mode configuration in the ModuleConfig `deckhouse`. If you're using a registry other than `registry.deckhouse.io`, refer to the `deckhouse` module documentation for correct configuration.

Configuration example:

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

5. Check the registry switch status in the `registry-state` secret using this guide.

   Example output:

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

6. If you need to switch back to the old registry management method, refer to the instruction.

⚠ This is a deprecated format for registry management.

Migration to the registry module

During the migration, Containerd v1 will switch to the new registry configuration format. Containerd v2 uses the new format by default. For more details, see the section with a description of configuration methods.

For containerd v2

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

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

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

   Specify this configuration when setting up the `Unmanaged` mode:

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

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

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

For Containerd v1

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

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

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

    REGISTRY_URL=private.registry.example

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

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

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

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

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

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

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

   Specify this configuration when setting up the `Unmanaged` mode:

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

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

   Example output:

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

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

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

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

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

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

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

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

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

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

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

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

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

```
d8 k get nodegroupconfiguration
```

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

Migration back from the registry module

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

1. Switch the registry to `Unmanaged` mode. If you are using a registry other than `registry.deckhouse.io`, refer to the

`deckhouse` module documentation for proper configuration.

Example configuration:

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

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

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

3. Switch the registry to the non-configurable `Unmanaged` mode. Example configuration:

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

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

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

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

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

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
              endpoint = ["https://${REGISTRY_URL}"]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]
            [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
              username = "username"
              password = "password"
              # OR
              auth = "dXNlcm5hbWU6cGFzc3dvcmQ="
    EOF
```

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

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

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

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

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

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

Check registry mode switch status

The status of the registry mode switch can be retrieved using the following command:

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

Example output:

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

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

Description of conditions:

Condition	Description
`ContainerdConfigPreflightReady`	State of the containerd configuration preflight check. Verifies there are no custom containerd auth configurations on the nodes.
`TransitionContainerdConfigReady`	State of preparing the containerd configuration for the new mode. Verifies that the configuration contains both the old and new mode settings.

Condition	Description
`FinalContainerdConfigReady`	State of finalizing the switch to the new containerd mode. Verifies that the containerd configuration has been successfully applied and contains only the new mode settings.
`DeckhouseRegistrySwitchReady`	State of switching Deckhouse and its components to use the new registry. `True` means Deckhouse successfully switched and is ready to operate.
`InClusterProxyReady`	State of In-Cluster Proxy readiness. Checks that the In-Cluster Proxy has started successfully and is running.
`CleanupInClusterProxy`	State of cleaning up the In-Cluster Proxy if it is not needed in the selected mode. Verifies that all related resources have been removed.
`Ready`	Overall state of registry readiness in the selected mode. Indicates that all other conditions are met and the module is ready to operate.

Additional resources

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

Restoring registry access when license token is expired or invalid

⚠ Use this guide only when the standard registry change procedure is unavailable.

If, after the license token expired, the Deckhouse Kubernetes Platform (DKP) pods were restarted, their logs will show a registry connection error when pulling DKP images. To switch the cluster to a new token, run the following steps on any master node:

1. Save the current `deckhouse-registry` secret configuration to a temporary file:

   ```
   d8 k -n d8-system get secret deckhouse-registry -o yaml > /tmp/deckhouse-registry.yaml
   ```

2. In the temporary file `/tmp/deckhouse-registry.yaml`, replace the `.dockerconfigjson` field with a Base64-encoded string containing the registry connection parameters. You can generate the required string with the commands below, substituting your own `MYPASSWORD` and `MYREGISTRY` values:

   ```
   declare MYUSER='license-token'
   declare MYPASSWORD='example-token'
   declare MYREGISTRY='example-regsitry.deckhouse.ru'
   MYAUTH=$(echo -n "$MYUSER:$MYPASSWORD" | base64 -w0)
   MYRESULTSTRING=$(echo -n "{\"auths\":{\"$MYREGISTRY\":
   {\"username\":\"$MYUSER\",\"password\":\"$MYPASSWORD\",\"auth\":\"$MYAUTH\"}}}" | base64 -w0)
   echo "$MYRESULTSTRING"
   ```

3. Allow updating the stale secret:

   ```
   d8 k delete validatingadmissionpolicybindings.admissionregistration.k8s.io heritage-label-objects.deckhouse.io
   ```

4. Import the updated configuration:

   ```
   d8 k -n d8-system apply -f /tmp/deckhouse-registry.yaml
   ```

5. Find the problematic `deckhouse` Pod on the current master node and delete it:

   ```
   d8 k get pods -n d8-system -o wide
   d8 k delete pod -n d8-system deckhouse-<id>
   ```

6. Make sure the new `deckhouse` Pod has started successfully:

   ```
   d8 k get pods -n d8-system
   ```

7. If necessary, delete any remaining `deckhouse` Pods that are in an incorrect state.

8. Repeat the standard procedure for changing the registry, substituting your token, the required registry address, and the edition instead of `example`:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller helper change-registry --user
license-token --password MY-PASSWORD registry-example.deckhouse.ru/deckhouse/example
```

Switching editions

Switching DKP to CE/BE/SE/SE+/EE

Switching DKP to CE/BE/SE/SE+/EE can be done in one of the following ways:

- using the `registry` module;
- without using the `registry` module.

⚠
- The functionality of this guide is validated for Deckhouse versions starting from `v1.70`. If your version is older, use the corresponding documentation.
- For commercial editions, you need a valid license key that supports the desired edition. If necessary, you can request a temporary key.
- The guide assumes the use of the public container registry address: `registry.deckhouse.io`. If you are using a different container registry address, modify the commands accordingly or refer to the guide on switching Deckhouse to use a different registry.
- The Deckhouse CE/BE/SE/SE+ editions do not support the cloud providers `dynamix`, `openstack`, `VCD`, and `vSphere` (vSphere is supported in SE+) and a number of modules.
- All commands are executed on the master node of the existing cluster with `root` user.
- If, during the switching process, the image of a module did not reload and the module did not reinstall, use the instructions to resolve the issue.

Switching using the registry module

1. Make sure the cluster has been migrated to be managed by the `registry` module.
   If the cluster is not managed by the `registry` module, proceed to the instruction.
2. Prepare variables for the license key and new edition name:

   i
   It is not necessary to fill the `NEW_EDITION` variable when switching to Deckhouse CE edition. The `NEW_EDITION` variable should match your desired Deckhouse edition. For example, to switch to:
   - CE, the variable should be `ce`;
   - BE, the variable should be `be`;
   - SE, the variable should be `se`;
   - SE+, the variable should be `se-plus`;
   - EE, the variable should be `ee`.

   ```
   NEW_EDITION=<PUT_YOUR_EDITION_HERE>
   LICENSE_TOKEN=<PUT_YOUR_LICENSE_TOKEN_HERE>
   ```
3. Ensure the Deckhouse queue is empty and error-free:

   ```
   d8 k -n d8-system exec -it svc/deckhouse-leader -c deckhouse -- deckhouse-controller queue list
   ```

   Example of the output (queues are empty):

   ```
   Summary:
   - 'main' queue: empty.
   - 88 other queues (0 active, 88 empty): 0 tasks.
   - no tasks to handle.
   ```
4. Start a temporary pod for the new Deckhouse edition to obtain current digests and a list of modules:

   For the CE edition:

   ```
   DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
   '.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk -F: '{print
   $NF}')
   d8 k run $NEW_EDITION-image --
   image=registry.deckhouse.ru/deckhouse/$NEW_EDITION/install:$DECKHOUSE_VERSION --command sleep -- infinity
   ```

   For other editions:

```
d8 k create secret docker-registry $NEW_EDITION-image-pull-secret \
 --docker-server=registry.deckhouse.ru \
 --docker-username=license-token \
 --docker-password=${LICENSE_TOKEN}

DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk -F: '{print
$NF}')
d8 k run $NEW_EDITION-image \
 --image=registry.deckhouse.ru/deckhouse/$NEW_EDITION/install:$DECKHOUSE_VERSION \
 --overrides="{\"spec\": {\"imagePullSecrets\":[{\"name\": \"$NEW_EDITION-image-pull-
secret\"}]}}" \
 --command sleep -- infinity
```

Once the pod is in `Running` state, execute the following commands:

```
NEW_EDITION_MODULES=$(d8 k exec $NEW_EDITION-image -- ls -l deckhouse/modules/ | grep -oE "\d.*-\w*" |
awk {'print $9'} | cut -c5-)
USED_MODULES=$(d8 k get modules -o custom-
columns=NAME:.metadata.name,SOURCE:.properties.source,STATE:.properties.state,ENABLED:.status.phase | grep Embedded |
grep -E 'Enabled|Ready' | awk {'print $1'})
MODULES_WILL_DISABLE=$(echo $USED_MODULES | tr ' ' '\n' | grep -Fxv -f <(echo $NEW_EDITION_MODULES
| tr ' ' '\n'))
```

5. Verify that the modules used in the cluster are supported in the desired edition. To see the list of modules not supported in the new edition and will be disabled:

   ```
   echo $MODULES_WILL_DISABLE
   ```

   > **i** Check the list to ensure the functionality of these modules is not in use in your cluster and you are ready to disable them.

   Disable the modules not supported by the new edition:

   ```
   echo $MODULES_WILL_DISABLE | tr ' ' '\n' | awk {'print "d8 platform module disable",$1'} | bash
   ```

   Wait for the Deckhouse pod to reach `Ready` state and ensure all tasks in the queue are completed:

   ```
   d8 k -n d8-system exec -it svc/deckhouse-leader -c deckhouse -- deckhouse-controller queue list
   ```

   Example of the output (queues are empty):

   ```
   Summary:
   - 'main' queue: empty.
   - 88 other queues (0 active, 88 empty): 0 tasks.
   - no tasks to handle.
   ```

6. Delete the created Secret and Pod:

   ```
   d8 k delete pod/$NEW_EDITION-image
   d8 k delete secret/$NEW_EDITION-image-pull-secret
   ```

7. Perform the switch to the new edition. To do this, specify the following parameter in the `deckhouse` ModuleConfig. For detailed configuration, refer to the `deckhouse` module documentation.

```
---
# Example for Direct mode
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Direct
      direct:
        # Relax mode is used to check for the presence of the current Deckhouse version in the
specified registry.
        # This mode must be used to switch between editions.
        checkMode: Relax
        # Specify your value for <NEW_EDITION>.
        imagesRepo: registry.deckhouse.ru/deckhouse/<NEW_EDITION>
        scheme: HTTPS
        # Specify your value for <LICENSE_TOKEN>.
        # If switching to the CE edition, remove this parameter.
        license: <LICENSE_TOKEN>
---
# Example for Unmanaged mode.
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  enabled: true
  settings:
    registry:
      mode: Unmanaged
      unmanaged:
        # Relax mode is used to check for the presence of the current Deckhouse version in the
specified registry.
        # This mode must be used to switch between editions.
        checkMode: Relax
        # Specify your value for <NEW_EDITION>.
        imagesRepo: registry.deckhouse.ru/deckhouse/<NEW_EDITION>
        scheme: HTTPS
        # Specify your value for <LICENSE_TOKEN>.
        # If switching to the CE edition, remove this parameter.
        license: <LICENSE_TOKEN>
```

8. Wait for the registry to switch. To verify the switch progress, follow the instruction.

   Example output:

   ```
   conditions:
     - lastTransitionTime: "..."
       message: |-
         Mode: Relax
         registry.deckhouse.ru: all 1 items are checked
       reason: Ready
       status: "True"
       type: RegistryContainsRequiredImages
   # ...
     - lastTransitionTime: "..."
       message: ""
       reason: ""
       status: "True"
       type: Ready
   ```

9. After the switch, remove the `checkMode: Relax` parameter from the `deckhouse` ModuleConfig to revert to the default check mode.
   Removing this parameter will trigger a check for the presence of critical components in the registry.

10. Wait for the check to complete by following the instruction.

    Example output:

```
conditions:
  - lastTransitionTime: "..."
    message: |-
      Mode: Default
      registry.deckhouse.ru: all 155 items are checked
    reason: Ready
    status: "True"
    type: RegistryContainsRequiredImages
# ...
  - lastTransitionTime: "..."
    message: ""
    reason: ""
    status: "True"
    type: Ready
```

11. Check for any pods in the `d8-*` namespaces that are in an error state and unable to load images. This must be done manually, as Deckhouse modules currently do not automatically reinitialize after the changes described above.

    Get a list of pods:

    ```
    d8 k get po -A
    ```

    Get detailed information about problematic pods:

    ```
    d8 k describe po <pod_name> <namespace>
    ```

    Re-download the modules corresponding to the problematic pods by running the following command on all master nodes:

    ```
    rm -rf /var/lib/deckhouse/downloaded/<module-name>/
    ```

    To get `<module-name>`, run the command:

    ```
    d8 k get modules
    ```

    After removing the data of the required modules, restart Deckhouse:

    ```
    d8 k rollout restart deploy -n d8-system deckhouse
    ```

12. Check if there are any pods with the Deckhouse old edition address left in the cluster, where `<YOUR-PREVIOUS-EDITION>` is your previous edition name:

    For Unmanaged mode:

    ```
    d8 k get pods -A -o json | jq -r '.items[] | select(.spec.containers[] | select(.image |
    contains("deckhouse.ru/deckhouse/<YOUR-PREVIOUS-EDITION>"))) | .metadata.namespace + "\t" +
    .metadata.name' | sort | uniq
    ```

    For other modes that use a fixed registry address.
    This check does not take external modules into account:

    ```
    # Get the list of valid digest values from the images_digests.json file inside Deckhouse.
    IMAGES_DIGESTS=$(d8 k -n d8-system exec -i svc/deckhouse-leader -c deckhouse -- cat
    /deckhouse/modules/images_digests.json | jq -r '.[][]' | sort -u)

    # Check for Pods using Deckhouse images from `registry.d8-system.svc:5001/system/deckhouse`
    # with a digest that is NOT present in the list of valid digest values (IMAGES_DIGESTS).
    d8 k get pods -A -o json |
    jq -r --argjson digests "$(printf '%s\n' $IMAGES_DIGESTS | jq -R . | jq -s .)" '
      .items[]
      | {name: .metadata.name, namespace: .metadata.namespace, containers: .spec.containers}
      | select(.containers != null)
      | select(
          .containers[]
          | select(.image | test("registry.d8-system.svc:5001/system/deckhouse") and
    test("@sha256:"))
          | .image as $img
          | ($img | split("@") | last) as $digest
          | ($digest | IN($digests[]) | not)
        )
      | .namespace + "\t" + .name
    ' | sort -u
    ```

Switching without using the registry module

1. If the `registry` module is enabled, disable it by following instruction.
2. Prepare variables for the license key and new edition name:

   > **i** It is not necessary to fill the `NEW_EDITION` and `AUTH_STRING` variables when switching to Deckhouse CE edition. The `NEW_EDITION` variable should match your desired Deckhouse edition. For example, to switch to:
   >
   > - CE, the variable should be `ce`;
   > - BE, the variable should be `be`;
   > - SE, the variable should be `se`;
   > - SE+, the variable should be `se-plus`;
   > - EE, the variable should be `ee`.

   ```
   NEW_EDITION=<PUT_YOUR_EDITION_HERE>
   LICENSE_TOKEN=<PUT_YOUR_LICENSE_TOKEN_HERE>
   AUTH_STRING="$(echo -n license-token:${LICENSE_TOKEN} | base64 )"
   ```

3. Ensure the Deckhouse queue is empty and error-free:

   ```
   d8 k -n d8-system exec -it svc/deckhouse-leader -c deckhouse -- deckhouse-controller queue list
   ```

   Example of the output (queues are empty):

   ```
   Summary:
   - 'main' queue: empty.
   - 88 other queues (0 active, 88 empty): 0 tasks.
   - no tasks to handle.
   ```

4. Create a `NodeGroupConfiguration` resource for temporary authorization in `registry.deckhouse.io`:

   > **i** Before creating a resource, refer to the section How to add configuration for an additional registry
   >
   > Skip this step if switching to Deckhouse CE.

   ```
   d8 k apply -f - <<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: NodeGroupConfiguration
   metadata:
     name: containerd-$NEW_EDITION-config.sh
   spec:
     nodeGroups:
     - '*'
     bundles:
     - '*'
     weight: 30
     content: |
       _on_containerd_config_changed() {
         bb-flag-set containerd-need-restart
       }
       bb-event-on 'containerd-config-file-changed' '_on_containerd_config_changed'
       mkdir -p /etc/containerd/conf.d
       bb-sync-file /etc/containerd/conf.d/$NEW_EDITION-registry.toml - containerd-config-file-
   changed << "EOF_TOML"
       [plugins]
         [plugins."io.containerd.grpc.v1.cri"]
           [plugins."io.containerd.grpc.v1.cri".registry.configs]
             [plugins."io.containerd.grpc.v1.cri".registry.configs."registry.deckhouse.io".auth]
               auth = "$AUTH_STRING"
       EOF_TOML
   EOF
   ```

   Wait for the `/etc/containerd/conf.d/$NEW_EDITION-registry.toml` file to appear on the nodes and for bashible synchronization to complete. To track the synchronization status, check the `UPTODATE` value (the number of nodes in this status should match the total number of nodes (`NODES`) in the group):

   ```
   d8 k get ng -o custom-columns=NAME:.metadata.name,NODES:.status.nodes,READY:.status.ready,UPTODATE:.status.upToDate -w
   ```

   Example output:

```
NAME     NODES   READY   UPTODATE
master   1       1       1
worker   2       2       2
```

Also, a message stating `Configuration is in sync, nothing to do` should appear in the systemd service log for bashible by executing the following command:

```
journalctl -u bashible -n 5
```

Example output:

```
Aug 21 11:04:28 master-ee-to-se-0 bashible.sh[53407]: Configuration is in sync, nothing to do.
Aug 21 11:04:28 master-ee-to-se-0 bashible.sh[53407]: Annotate node master-ee-to-se-0 with
annotation node.deckhouse.io/configuration-
checksum=9cbe6db6c91574b8b732108a654c99423733b20f04848d0b4e1e2dadb231206a
Aug 21 11:04:29 master ee-to-se-0 bashible.sh[53407]: Successful annotate node master-ee-to-se-
0 with annotation node.deckhouse.io/configuration-
checksum=9cbe6db6c91574b8b732108a654c99423733b20f04848d0b4e1e2dadb231206a
Aug 21 11:04:29 master-ee-to-se-0 systemd[1]: bashible.service: Deactivated successfully.
```

5. Start a temporary pod for the new Deckhouse edition to obtain current digests and a list of modules:

```
DECKHOUSE_VERSION=$(d8 k -n d8-system get deploy deckhouse -ojson | jq -r
'.spec.template.spec.containers[] | select(.name == "deckhouse") | .image' | awk -F: '{print
$NF}')
d8 k run $NEW_EDITION-image --
image=registry.deckhouse.io/deckhouse/$NEW_EDITION/install:$DECKHOUSE_VERSION --command sleep --infinity
```

6. Once the pod is in `Running` state, execute the following commands:

```
NEW_EDITION_MODULES=$(d8 k exec $NEW_EDITION-image -- ls -l deckhouse/modules/ | grep -oE "\d.*-\w*" |
awk {'print $9'} | cut -c5-)
USED_MODULES=$(d8 k get modules -o custom-
columns=NAME:.metadata.name,SOURCE:.properties.source,STATE:.properties.state,ENABLED:.status.phase | grep Embedded |
grep -E 'Enabled|Ready' | awk {'print $1'})
MODULES_WILL_DISABLE=$(echo $USED_MODULES | tr ' ' '\n' | grep -Fxv -f <(echo $NEW_EDITION_MODULES
| tr ' ' '\n'))
```

7. Verify that the modules used in the cluster are supported in the desired edition. To see the list of modules not supported in the new edition and will be disabled:

```
echo $MODULES_WILL_DISABLE
```

> **i** Check the list to ensure the functionality of these modules is not in use in your cluster and you are ready to disable them.

Disable the modules not supported by the new edition:

```
echo $MODULES_WILL_DISABLE | tr ' ' '\n' | awk {'print "d8 platform module disable",$1'} | bash
```

Wait for the Deckhouse pod to reach `Ready` state and ensure all tasks in the queue are completed:

```
d8 k -n d8-system exec -it svc/deckhouse-leader -c deckhouse -- deckhouse-controller queue list
```

Example of the output (queues are empty):

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

8. Execute the `deckhouse-controller helper change-registry` command from the Deckhouse pod with the new edition parameters:

   To switch to BE/SE/SE+/EE editions:

```
DOCKER_CONFIG_JSON=$(echo -n "{\"auths\": {\"registry.deckhouse.io\": {\"username\": \"license-token\", \"password\": \"${LICENSE_TOKEN}\", \"auth\": \"${AUTH_STRING}\"}}}" | base64 -w 0)
d8 k --as system:sudouser -n d8-cloud-instance-manager patch secret deckhouse-registry --type merge --patch="{\"data\":{\".dockerconfigjson\":\"$DOCKER_CONFIG_JSON\"}}"
d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller helper change-registry --user=license-token --password=$LICENSE_TOKEN --new-deckhouse-tag=$DECKHOUSE_VERSION registry.deckhouse.io/deckhouse/$NEW_EDITION
```

To switch to CE edition:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller helper change-registry --new-deckhouse-tag=$DECKHOUSE_VERSION registry.deckhouse.io/deckhouse/ce
```

9. Check if there are any pods with the Deckhouse old edition address left in the cluster, where `<YOUR-PREVIOUS-EDITION>` your previous edition name:

```
d8 k get pods -A -o json | jq -r '.items[] | select(.spec.containers[] | select(.image | contains("deckhouse.io/deckhouse/<YOUR-PREVIOUS-EDITION>"))) | .metadata.namespace + "\t" + .metadata.name' | sort | uniq
```

10. Delete temporary files, the `NodeGroupConfiguration` resource, and variables:

> **i** Skip this step if switching to Deckhouse CE.

```
d8 k delete ngc containerd-$NEW_EDITION-config.sh
d8 k delete pod $NEW_EDITION-image
d8 k apply -f - <<EOF
    apiVersion: deckhouse.io/v1alpha1
    kind: NodeGroupConfiguration
    metadata:
      name: del-temp-config.sh
    spec:
      nodeGroups:
      - '*'
      bundles:
      - '*'
      weight: 90
      content: |
        if [ -f /etc/containerd/conf.d/$NEW_EDITION-registry.toml ]; then
          rm -f /etc/containerd/conf.d/$NEW_EDITION-registry.toml
        fi
EOF
```

After the bashible synchronization completes (synchronization status on the nodes is shown by the `UPTODATE` value in NodeGroup), delete the created NodeGroupConfiguration resource:

```
d8 k delete ngc del-temp-config.sh
```

Additional resources

- Module `deckhouse` documentation
- Module `node-manager` documentation
- Module `registry` documentation

Authentication

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

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

Depending on the configuration, DKP supports two authentication approaches:

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

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

The platform also provides capabilities for:

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

Additional resources

- Module `user-authn` custom resources

Integration with external authentication providers

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

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

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

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

General integration workflow

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

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

   > **i** **Important**: When specifying the redirect URI, substitute the actual value of `publicDomainTemplate` without `%s`.
   > For example, if `publicDomainTemplate: '%s.sandbox1.deckhouse-docs.flant.com'`, the actual URI would be:
   > `https://dex.sandbox20.deckhouse-docs.flant.com/callback`.
   >
   > To retrieve the Dex address (URI), run:
   >
   > ```
   > d8 k -n d8-user-authn get ingress dex -o jsonpath="{.spec.rules[*].host}"
   > ```

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

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

   Check the module status:

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

   Example output:

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

   Enable the module via CL:

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

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

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

   - Specify the required module parameters in the `spec.settings` section. For more details about the `user-`

`authn` module settings, refer to the module reference.

Example `user-authn` configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    kubeconfigGenerator:
    - id: direct
      masterURI: https://159.89.5.247:6443
      description: "Direct access to kubernetes API"
    publishAPI:
      enabled: true
```

After setting up integration in the cluster with an external authentication provider, authentication through it will become possible in the platform's web interfaces. Information on setting up authentication for user applications can be found in the section Usage → IAM → Authentication.

OIDC (OpenID Connect) integration

Authentication via an OIDC provider requires registering a client (or creating an application). Follow your provider's documentation to do this (e.g., Okta, Keycloak, Gluu, or Blitz).

Specify the `clientID` and `clientSecret` obtained during setup in the DexProvider resource.

> **i** When registering an application with any OIDC provider, you must specify a redirect URI.
> For integration with DexProvider, use the following format: `https://dex.<publicDomainTemplate>/callback`, where `publicDomainTemplate` is the DNS name template of your cluster as defined in the `global` module.

> **i** To ensure proper token revocation on logout and to force re-authentication, set the `prompt` parameter to `login`. This guarantees that the user will be prompted to re-enter credentials during subsequent logins.

To configure fine-grained access control for users in applications:

- Add the `allowedUserGroups` parameter to the ModuleConfig of the target application.
- Assign the appropriate groups to the user, using identical group names on both the provider and Deckhouse sides.

Keycloak

During Keycloak configuration, select the appropriate `realm`, add a user in the Users section, and create a client in the Clients section with authentication enabled, which is required to generate a `clientSecret`. Then follow these steps:

1. In the Client scopes section, create a `scope` named `groups` and assign it a mapper `Group Membership` ("Client scopes" → "Client scope details" → "Mappers" → "Configure a new mapper"). Set values of "Name" and "Token Claim Name" as `groups` and turn off "Full group path".
2. In the previously created client, add this `scope` in the Client scopes tab ("Clients" → "Client details" → "Client Scopes" → "Add client scope").
3. In the "Valid redirect URIs", "Valid post logout redirect URIs", and "Web origins" fields in the client configuration, specify `https://dex.<publicDomainTemplate>/*`, where `publicDomainTemplate` is the cluster DNS name template defined in the `global` module.

Example provider configuration for Keycloak integration:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: keycloak
spec:
  type: OIDC
  displayName: My Company Keycloak
  oidc:
    issuer: https://keycloak.my-company.com/realms/myrealm # Use your realm name
    clientID: plainstring
    clientSecret: plainstring
    insecureSkipEmailVerified: true
    getUserInfo: true
    scopes:
      - openid
      - profile
      - email
      - groups
```

If email verification is not enabled in Keycloak, to properly use it as an identity provider, adjust the `Client Scopes` settings in one of the following ways:

- Delete the `Email verified` mapping (“Client Scopes” → “Email” → “Mappers”). This is required for proper processing of the `insecureSkipEmailVerified` field when it’s set to `true` and for correct permission assignment to users with unverified emails.
- If you can’t modify or delete the `Email verified` mapping, create a new Client Scope named `email_dkp` (or any other name) and add two mappings:
  - `email`: “Client Scopes” → `email_dkp` → “Add mapper” → “From predefined mappers” → `email`.
  - `email verified`: “Client Scopes” → `email_dkp` → “Add mapper” → “By configuration” → “Hardcoded claim”. Specify the following fields:
    - “Name”: `email verified`
    - “Token Claim Name”: `emailVerified`
    - “Claim value”: `true`
    - “Claim JSON Type”: `boolean`

  After that, in the client registered for the DKP cluster in “Clients”, change `Client scopes` from `email` to `email_dkp`.

  In the DexProvider resource, specify `insecureSkipEmailVerified: true` and in the `.spec.oidc.scopes` field, change the Client Scope name to `email_dkp` following the example:

  ```
  scopes:
   - openid
   - profile
   - email_dkp
   - groups
  ```

Blitz Identity Provider

Example configuration of a provider for integration with Blitz Identity Provider:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: blitz
spec:
  displayName: Blitz Identity Provider
  oidc:
    basicAuthUnsupported: false
    claimMapping:
      email: email
      groups: your_claim # Claim for obtaining user groups; groups are configured on the Blitz
Identity Provider side.
    clientID: clientID
    clientSecret: clientSecret
    getUserInfo: true
    insecureSkipEmailVerified: true # Set to true if email verification is not required.
    insecureSkipVerify: false
    issuer: https://yourdomain.idblitz.ru/blitz
    promptType: consent
    scopes:
    - profile
    - openid
    userIDKey: sub
    userNameKey: email
  type: OIDC
```

Okta

Example configuration of a provider for integration with Okta:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: okta
spec:
  type: OIDC
  displayName: My Company Okta
  oidc:
    issuer: https://my-company.okta.com
    clientID: plainstring
    clientSecret: plainstring
    insecureSkipEmailVerified: true
    getUserInfo: true
```

After enabling integration with Okta, you can use Okta user groups to manage access rights. For example, you can specify a list of groups whose members are allowed to access Grafana.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  settings:
    auth:
      allowedUserGroups:
        - adm-grafana-access
        - grafana-access
```

SAML 2.0 integration

The example shows the provider’s settings for integration with a SAML 2.0 Identity Provider (e.g., AD FS, Okta, Keycloak).

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: saml-provider
spec:
  type: SAML
  displayName: Corporate SAML
  saml:
    ssoURL: https://saml-idp.example.com/saml/sso
    rootCAData: |
      -----BEGIN CERTIFICATE-----
      MIIFaDC...
      -----END CERTIFICATE-----
    entityIssuer: https://dex.example.com/callback
    ssoIssuer: https://saml-idp.example.com
    usernameAttr: name
    emailAttr: email
    groupsAttr: groups
    nameIDPolicyFormat: persistent
```

To configure the SAML Identity Provider:

1. Register Dex as a Service Provider (SP) in your IdP with the following settings:
   - **ACS URL (Assertion Consumer Service)**: `https://dex.<modules.publicDomainTemplate>/callback`
   - **Entity ID**: `https://dex.<modules.publicDomainTemplate>/callback`
   - **NameID format**: `persistent` or `emailAddress`
2. Configure attribute mappings in the IdP to send `email`, `name` (username), and `groups` attributes in the SAML assertion.
3. Export the IdP signing certificate and specify it in the `rootCAData` field of the DexProvider resource.

> **i** SAML does not natively support refresh tokens. Dex caches the user identity from the initial SAML assertion and returns it on subsequent refresh requests. The session lifetime is controlled by the `expiry.refreshTokens` settings in the `user-authn` module configuration.

LDAP integration

To configure authentication, create a read-only account (service account) in your LDAP directory.
This account will be used to perform search queries in the LDAP catalog.

In the DexProvider resource, specify the following parameters:

- `bindDN`: Full Distinguished Name (DN) of the created service account. For example: `cn=readonly,dc=example,dc=org`.
- `bindPW`: Password for the specified `bindDN`.

> **i** If your LDAP server allows anonymous access for search queries, the `bindDN` and `bindPW` parameters can be omitted.
> However, using authenticated access is recommended for improved security.
>
> The `bindPW` parameter must contain the password in plain text. Dex does not support hashed passwords in this field.

You can also enable support for:

- Basic Authentication for accessing the Kubernetes API using LDAP credentials. To do this, set the `enableBasicAuth: true` parameter in the DexProvider resource. For more information, see Access using Basic Authentication.
- Kerberos SSO (SPNEGO) for LDAP. To do this, enable the `spec.ldap.kerberos` block in the DexProvider resource and configure the necessary parameters in it. For more details, see Kerberos SSO (SPNEGO) for LDAP.

Example configuration for integrating with Active Directory (Basic Authentication):

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: active-directory
spec:
  type: LDAP
  displayName: Active Directory
  ldap:
    host: ad.example.com:636
    insecureSkipVerify: true

    bindDN: cn=Administrator,cn=users,dc=example,dc=com
    bindPW: admin0!

    enableBasicAuth: true

    usernamePrompt: Email Address

    userSearch:
      baseDN: cn=Users,dc=example,dc=com
      filter: "(objectClass=person)"
      username: userPrincipalName
      idAttr: DN
      emailAttr: userPrincipalName
      nameAttr: cn

    groupSearch:
      baseDN: cn=Users,dc=example,dc=com
      filter: "(objectClass=group)"
      userMatchers:
      - userAttr: DN
        groupAttr: member
      nameAttr: cn
```

GitHub integration

You need to create a new application in your GitHub organization.

To do this, follow these steps:

1. Go to “Settings → Developer settings → OAuth Apps → New OAuth App”, and set the “Authorization callback URL” to `https://dex.<publicDomainTemplate>/callback`.
2. Use the generated `Client ID` and `Client Secret` in the DexProvider resource.

If the GitHub organization is managed by a client:

1. Go to “Settings → Applications → Authorized OAuth Apps → `<name of created OAuth App>`” and click “Send Request” to submit an approval request.
2. Ask the client to approve the request that will be sent to their email.

Example configuration for integrating with GitHub:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: github
spec:
  type: Github
  displayName: My Company GitHub
  # Optional: disable the provider without deleting CR
  # enabled: false
  github:
    clientID: plainstring
    clientSecret: plainstring
```

GitLab integration

You need to create a new application in your GitLab project.

To do this, follow these steps:

1. For self-hosted GitLab: go to “Admin Area → Applications → New application” and set the “Redirect URI (Callback URL)” to `https://dex.<publicDomainTemplate>/callback`. Also, select the following scopes: `read_user`, `openid`.
2. For GitLab Cloud (gitlab.com): under the main account of the project, go to “User Settings → Applications → Add new application”, set the “Redirect URI (Callback URL)” to `https://dex.<publicDomainTemplate>/callback`, and select the **scopes**: `read user`, `openid`.
3. Use the obtained `Application ID` and secret in the DexProvider resource.

> **i** For GitLab version 16 and above, enable the “Trusted” option when creating the application.
> This option is available under “Admin Area → Applications”. Marking the app as trusted allows skipping the authorization step for users, which can be useful in controlled environments.

Example configuration for integrating with GitLab:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: gitlab
spec:
  type: Gitlab
  displayName: Dedicated Gitlab
  gitlab:
    baseURL: https://gitlab.example.com
    clientID: plainstring
    clientSecret: plainstring
    groups:
    - administrators
    - users
```

Atlassian Crowd integration

In the relevant Atlassian Crowd project, you need to create a new Generic application.

To do this, follow these steps:

1. Go to “Applications → Add application”.
2. Specify the obtained “Application Name” and “Password” in the DexProvider resource.
3. When specifying groups in the DexProvider resource, make sure their names are written in lowercase.
   This is necessary for correct group matching between Crowd and Deckhouse.

Example configuration for integrating with Atlassian Crowd:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: crowd
spec:
  type: Crowd
  displayName: Crowd
  crowd:
    baseURL: https://crowd.example.com/crowd
    clientID: plainstring
    clientSecret: plainstring
    enableBasicAuth: true
    groups:
    - administrators
    - users
```

Bitbucket Cloud Integration

To configure authentication, you need to create a new OAuth consumer in the Bitbucket team menu.

Follow these steps:

1. Go to “Personal settings → Access management → OAuth consumers → Add consumer”, and specify `https://dex.<publicDomainTemplate>/callback` as the “Callback URL”.
2. Grant access:
   - “Account: Read” → allows retrieval of basic user information (e.g., email, username).
   - “Workspace membership: Read” → allows retrieval of user workspace membership information.
3. Use the obtained `Key` and secret in the DexProvider resource.

Example configuration for integrating with Bitbucket:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: bitbucket
spec:
  type: BitbucketCloud
  displayName: Bitbucket
  bitbucketCloud:
    clientID: plainstring
    clientSecret: plainstring
    includeTeamGroups: true
    teams:
    - administrators
    - users
```

Additional resources

- Global parameters
- Module `user-authn` documentation

Local authentication

In addition to external authentication providers, DKP also supports local authentication.

Local authentication provides user verification and access management with support for configurable password policies, two-factor authentication (2FA), and group management. The implementation complies with OWASP recommendations, ensuring reliable protection of access to the cluster and applications without requiring integration with external authentication systems.

Local authentication involves creating User and Group resources in the cluster for static users and groups:

- A User object stores user information, including email and a hashed password (the password is not stored in plain text).
- A Group object defines a list of users grouped together.

Creating a static user

To create a static user, create a User resource.

Example resource definition (note that the example includes a ttl):

```
apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  email: admin@yourcompany.com
  password: $2a$10$etblbZ9yfZaKgbvysf1qguW3WULdMnxwWFrkoKpRH1yeWa5etjjAa
  ttl: 24h
```

Come up with a password and specify it in the `password` field as a raw bcrypt hash or a Base64-encoded bcrypt hash. To generate a Base64-encoded bcrypt hash, use the following command:

```
echo -n '<PASSWORD>' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0; echo
```

> **i** If the `htpasswd` command is not available, install the appropriate package:
>
> - `apache2-utils` — for Debian-based distributions.
> - `httpd-tools` — for CentOS-based distributions.
> - `apache2-htpasswd` — for ALT Linux.

Local user operations

Password reset, 2FA reset, and user lock/unlock operations are performed via the UserOperation resource. The `initiatorType` field indicates who initiated the operation: an administrator (`admin`), the system (`system`), or the user (`self`).

Administrative operations

Use the `d8 iam user` commands for administrative actions on local users. They create a UserOperation resource with `initiatorType: admin`, wait for the operation to complete, and print the result.

The `ResetPassword`, `Reset2FA`, and `Lock` operations delete the user's Dex OfflineSessions and RefreshToken objects. This terminates the user's active offline sessions and requires re-authentication.

Examples of using the `d8 iam user` commands:

- Interactive password reset:

```
d8 iam user reset-password admin
```

- Reading the new password from stdin:

```
echo "N3wPa$$wo#d" | d8 iam user reset-password admin --password-stdin
```

- Generating a new password automatically:

```
d8 iam user reset-password admin --generate-password
```

- Reset the password in hashed form (if the password is hashed, provide the bcrypt hash without Base64 encoding):

```
d8 iam user reset-password admin --password-hash '$2y$10$abcdef...'
```

- 2FA reset:

```
d8 iam user reset2fa admin
```

- Locking a user for 30 minutes:

```
d8 iam user lock admin 30m
```

- User unlock:

```
d8 iam user unlock admin
```

By default, commands wait for the operation to complete. To only create a UserOperation and print its name, use the `--wait=false` flag.

Self-service password reset

A local user can reset their own password in the DKP authentication interface. This creates a UserOperation resource with `type: ResetPassword` and `initiatorType: self`.

Self-service password reset is available only for local accounts (the built-in `Local` connector). Users who sign in through external authentication providers must contact the administrator of the corresponding system.

When a user resets their password:

- The new password must comply with the password policy.
- The user's active sessions are terminated and re-authentication is required.

For user-facing password change and reset scenarios, see Configuring authentication for applications.

Adding a user to a group

> ⚠ It is forbidden to use users and groups with the `system:` prefix.
> Authentication attempts by such users or members of such groups will be rejected, and a corresponding warning will appear in the `kube-apiserver` logs.

To group static users together, create a Group resource.

Example resource definition:

```
apiVersion: deckhouse.io/v1alpha1
kind: Group
metadata:
  name: admins
spec:
  name: admins
  members:
    - kind: User
      name: admin
```

Where `members` is a list of users belonging to the group.

Once the group is created and includes all necessary users, proceed by configuring authorization.

Configuring password policy

Password policy allows controlling password complexity, rotation, and user lockout.

To set up a password policy, use the `passwordPolicy` field in the configuration of the `user-authn` module:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    passwordPolicy:
      complexityLevel: Fair
      passwordHistoryLimit: 10
      lockout:
        lockDuration: 15m
        maxAttempts: 3
      rotation:
        interval: "30d"
```

Field description:

- `complexityLevel`: Password complexity level.
- `passwordHistoryLimit`: Number of previous passwords stored in the system to prevent their reuse.
- `lockout`: Lockout settings after exceeding the limit of failed login attempts:
  - `lockout.maxAttempts`: Limit of allowed failed login attempts.
  - `lockout.lockDuration`: User lockout duration.
- `rotation`: Password rotation settings:
  - `rotation.interval`: Period for mandatory password change.

Configuring two-factor authentication (2FA)

2FA increases security by requiring a code from a TOTP authenticator application (for example, Google Authenticator) during login.

To set up 2FA, use the `staticUsers2FA` field in the configuration of the `user-authn` module:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    staticUsers2FA:
      enabled: true
      issuerName: "awesome-app"
```

Field description:

- `enabled`: Enables or disables 2FA for all static users.
- `issuerName`: Name displayed in the authenticator application when adding an account.

> **i** After enabling 2FA, each user must register in the authenticator application during their first login.

Additional resources

- Module `user-authn` custom resources
- Module `user-authn` parameters

Accessing the Kubernetes API via load balancer

DKP allows using authentication when accessing the Kubernetes API. In this case, a user can generate a `kubectl` configuration via the DKP kubeconfig web interface to securely access the Kubernetes API through a traffic balancer (Ingress controller).

To configure access, follow these steps:

1. Enable Kubernetes API publishing. To do this, set the parameter `publishAPI.enabled: true` in the `user-authn` module settings or via the Deckhouse admin web interface.

   Example module configuration:

```
spec:
  enabled: true
  version: 2
  settings:
    publishAPI:
      enabled: true
```

2. Open the kubeconfig web interface.
   The kubeconfig generation interface in DKP is automatically activated after enabling the `publishAPI` parameter in the `user-authn` module.
   This web interface is available at the following URL:

   ```
   https://kubeconfig.<publicDomainTemplate>
   ```

   For example, if `publicDomainTemplate` is `%s.kube.my`, the URL will be `https://kubeconfig.kube.my`.

3. Generate the `kubectl` configuration.
   After logging into the kubeconfig interface, the user will receive a set of commands to configure `kubectl`.
   These commands can be copied and pasted into the terminal.
   Authentication will be performed using an OIDC token issued by Dex.
   If the provider supports session renewal, the configuration will include a `refresh token`, allowing access to be extended without re-authentication.

4. Configure multiple API access points.
   In the `user-authn` module configuration, you can specify multiple connection points (kube-apiserver), each with its own description and CA certificates.
   This may be useful if the cluster is accessible through different networks — for example, via VPN or public IP:

   ```
   settings:
     kubeconfigGenerator:
     - id: direct
       masterURI: https://159.89.5.247:6443
       description: "Direct access to kubernetes API"
   ```

How API access protection works in Kubernetes

In Deckhouse Kubernetes Platform, you can safely expose the Kubernetes API externally using an Ingress controller while maintaining access control. API exposure and authentication configuration are handled via the `user-authn` module. You can configure:

- A list of trusted IP addresses or networks allowed to access the API.
- A list of user groups permitted to authenticate.
- The Ingress controller through which access will be provided.

To configure:

1. Enable API publishing as shown in the example above.
2. Configure access restrictions. In the module configuration, you can specify:
   - A list of IP addresses or networks allowed to access (`allowedSourceRanges`).
   - A list of user groups allowed to connect to the Kubernetes API (`allowedUserGroups`).
   - The Ingress controller to be used for publishing (`ingressClass`).
3. Use the kubeconfig web interface.
   Users will be able to securely access the API using the kubeconfig generated via the web interface (`https://kubeconfig.<publicDomainTemplate>`).
   This kubeconfig will include the OIDC token and the Ingress connection settings.

The following will be automatically configured when API publishing is enabled:

- Deckhouse will set the required arguments for the kube-apiserver.
- A CA certificate will be generated and added to the kubeconfig.
- Login via Dex with OIDC support will be configured.

Access using Basic Authentication (LDAP)

In addition to OIDC, you can configure direct access to the API using Basic Authentication (username and password). In this case, credentials are verified against LDAP-compatible directory service.

To configure:

1. Enable API publishing (`publishAPI` parameter).
2. Configure an LDAP provider in the `user-authn` module and enable the `enableBasicAuth: true` option.

> ⚠ Only one provider in the cluster can have `enableBasicAuth` enabled.

After this, users can configure their `kubeconfig` by specifying their LDAP username and password:

```
apiVersion: v1
kind: Config
clusters:
- name: my-cluster
  cluster:
    server: https://api.example.com
    # Path to CA certificate or insecure-skip-tls-verify: true
    certificate-authority: /path/to/ca.crt
users:
- name: ldap-user
  user:
    username: janedoe@example.com
    password: userpassword
contexts:
- name: default
  context:
    cluster: my-cluster
    user: ldap-user
current-context: default
```

Kerberos (SPNEGO) SSO for LDAP

Dex supports passwordless Kerberos (SPNEGO) flow for the LDAP connector. The mechanism works as follows:

1. The browser, trusting the Dex host, sends `Authorization: Negotiate …`.
2. Dex validates the Kerberos ticket against the keytab and skips the login/password input form.
3. Dex matches the principal with the LDAP name, retrieves the groups, and completes the OIDC flow.

> **i** To configure Kerberos (SPNEGO) SSO, LDAP must have an account with read-only privileges (service account). For more information on configuration, see the section LDAP Integration.

Enabling Kerberos (SPNEGO) SSO for LDAP:

1. In AD/KDC, create/provision an SPN `HTTP/<dex-fqdn>` for a service account and generate a keytab.
2. In the cluster, create a Secret in the `d8-user-authn` namespace with the `krb5.keytab` data key.
3. In the LDAP DexProvider resource, enable `spec.ldap.kerberos`:
   - `enabled: true`
   - `keytabSecretName: <secret name>`
   - optional: `expectedRealm`, `usernameFromPrincipal`, `fallbackToPassword`

Dex will mount the keytab automatically and start accepting SPNEGO. A server-side `krb5.conf` is not required — tickets are validated using the keytab.

Example of configuring SSO via Kerberos (SPNEGO) for LDAP (extension of the LDAP provider specification):

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: active-directory
spec:
  type: LDAP
  displayName: Active Directory
  ldap:
    host: ad.example.com:636
    bindDN: cn=Administrator,cn=users,dc=example,dc=com
    bindPW: admin0!
    userSearch:
      baseDN: cn=Users,dc=example,dc=com
      username: sAMAccountName
      idAttr: uid
      emailAttr: mail
      nameAttr: cn
    groupSearch:
      baseDN: cn=Users,dc=example,dc=com
      nameAttr: cn
      userMatchers:
      - userAttr: uid
        groupAttr: memberUid
    kerberos:
      enabled: true
      keytabSecretName: dex-kerberos-keytab  # Secret in d8-user-authn with key 'krb5.keytab'.
      expectedRealm: EXAMPLE.COM           # Optional, case-insensitive match.
      usernameFromPrincipal: sAMAccountName  # localpart|sAMAccountName|userPrincipalName
      fallbackToPassword: false            # The default is false; if true, a login/password form is
rendered when the `Authorization: Negotiate` header is missing or invalid.
```

Notes:

- The Secret `dex-kerberos-keytab` must exist in the `d8-user-authn` namespace and have a data key named exactly `krb5.keytab`.
- A single Dex Pod can serve multiple LDAP+Kerberos providers. Each provider mounts its own keytab; a shared `krb5.conf` is not required (Dex validates tickets offline using the keytab).

Additional resources

- Module `user-authn` documentation

Resource configuration

Deckhouse Kubernetes Platform allows you to configure resource limits and requests for all components. By default, the following values are used:

- **Dex OIDC Provider** — 10m CPU / 25Mi memory (requests), 100m CPU / 100Mi memory (limits)
- **Kubeconfig Generator** — 10m CPU / 25Mi memory (requests), 100m CPU / 100Mi memory (limits)
- **Basic Auth Proxy** — 10m CPU / 25Mi memory (requests), 100m CPU / 100Mi memory (limits)
- **Dex Authenticator** — 10m CPU / 25Mi memory (requests), 100m CPU / 100Mi memory (limits)
- **Redis** (used by Dex Authenticator) — 10m CPU / 25Mi memory (requests), 100m CPU / 100Mi memory (limits)

> **i** When Vertical Pod Autoscaler (VPA) is enabled, the resource limits are managed automatically by VPA, but you can still configure the minimum and maximum allowed values through the `resources` section.

Example configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: v1
  settings:
    resources:
      dex:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
      kubeconfigGenerator:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
      basicAuthProxy:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
      dexAuthenticator:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
      redis:
        requests:
          cpu: "10m"
          memory: "25Mi"
        limits:
          cpu: "20m"
          memory: "50Mi"
```

Additional resources

- Module `vertical-pod-autoscaler` documentation

Authorization

In Deckhouse Kubernetes Platform (DKP), authorization is based on the standard Kubernetes Role-Based Access Control (RBAC) mechanism. This allows for flexible access control for different users, groups, and service accounts, ensuring security and operational control in the cluster.

DKP supports two role models:

- Current: The end-to-end authorization subsystem extends the standard RBAC mechanism using custom resources — ClusterAuthorizationRule and AuthorizationRule.
- Experimental: This model also relies on the standard RBAC mechanism. Access is configured by creating RoleBinding or ClusterRoleBinding resources.

Both models are supported by the `user-authz` module. The choice of model depends on security requirements and usage scenarios.

Who gets access and when

There are two main scenarios for granting access in DKP:

- Granting access to users for working via command-line clients, web UI, and other tools used for administration, development, and cluster management.
- Granting access to service accounts for automating tasks such as application deployment and updates (most commonly using the IaC approach). Examples of such services include CI/CD systems, monitoring systems, and others.

After successful authentication, users and service accounts are granted access to cluster resources based on the configured authorization settings.

User authentication

DKP supports multiple user authentication methods. For details, refer to User authentication.

Service account authentication

In Kubernetes, service accounts (ServiceAccount) are special accounts used to automate tasks and interact with the cluster API. They enable applications and services to securely communicate with the Kubernetes API. In DKP, service accounts for external services are created in the `d8-service-accounts` namespace to maintain consistency.

Example manifest for creating a ServiceAccount:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gitlab-runner-deploy
  namespace: d8-service-accounts
```

After creating the ServiceAccount, issue a token so the service can authenticate in the cluster. This is done by creating a secret that contains the access token.

Example manifest for creating a secret with a ServiceAccount token:

```
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-runner-deploy-token
  namespace: d8-service-accounts
  annotations:
    kubernetes.io/service-account.name: gitlab-runner-deploy
type: kubernetes.io/service-account-token
```

Additional resources

- Module `user-authz` documentation

Current authorization model

To use the current role model, the `user-authz` module must be enabled in the cluster. This module creates a set of cluster roles (ClusterRole) suitable for most user and group access management tasks.

> ⚠ Starting from Deckhouse Kubernetes Platform v1.64, the module includes an experimental role-based access model. The current role model will continue to function, but it will be deprecated in the future.
>
> The current and experimental role-based access models are incompatible. Automatic conversion of resources is not possible.

Key features of the current role model:

- Implements a role-based end-to-end authorization subsystem that extends the standard RBAC functionality.
- Access rights are managed using the custom resources ClusterAuthorizationRule and AuthorizationRule.
- Controls access to scaling tools (via the `allowScale` parameter in the ClusterAuthorizationRule or AuthorizationRule resource).
- Controls access to port forwarding (via the `portForwarding` parameter in the ClusterAuthorizationRule or AuthorizationRule resource).
- Manages the list of allowed namespaces using the labelSelector format (via the `namespaceSelector` parameter in the ClusterAuthorizationRule resource).

High-level roles used in the current model

In addition to RBAC, the current role model implemented via the `user-authz` module provides a convenient set of high-level roles:

Role	Examples of allowed actions	Limitations
User	View Pods, logs, and Deployments	Can’t access secrets, ports, or containers
PrivilegedUser	Access containers (via `kubectl exec`), view secrets, remove Pods	Can’t modify a Deployment or Service
Editor	Create and remove a Deployment, Service, or ConfigMap	Can’t access a ReplicaSet or ClusterRoles
Admin	Remove a ReplicaSet and control RBAC in a namespace	Can’t access cluster-wide resources
ClusterEditor	Create DaemonSet, ClusterRole, ClusterXXXMetric, KeepalivedInstance (only those required for applied tasks)	Can’t remove MachineSets

Role	Examples of allowed actions	Limitations
ClusterAdmin	Fully access ClusterRoleBindings, Machines, and OpenstackInstanceClasses	Can elevate their permissions
SuperAdmin	Any actions are allowed (including `*` in RBAC), but with consideration to `limitNamespaces`	Limitations can be applied only using cluster policies

⚠ The multitenancy mode (namespace-based authorization) is currently implemented as a temporary solution and **does not provide full isolation guarantees**.

If the `namespaceSelector` parameter is used in the ClusterAuthorizationRule resource, the `limitNamespaces` and `allowAccessToSystemNamespace` parameters are ignored.

If the webhook that implements the authorization system becomes unavailable for any reason, the `allowAccessToSystemNamespaces`, `namespaceSelector`, and `limitNamespaces` options in custom resources will no longer work, and users will have access to all namespaces. Once the webhook is available again, the options will resume functioning.

Default access list for each high-level role

Verb shortcuts:

- read: `get`, `list`, `watch`
- read-write: `get`, `list`, `watch`, `create`, `delete`, `deletecollection`, `patch`, `update`
- write: `create`, `delete`, `deletecollection`, `patch`, `update`

Role `User`:

```
read:
    - apiextensions.k8s.io/customresourcedefinitions
    - apps/daemonsets
    - apps/deployments
    - apps/replicasets
    - apps/statefulsets
    - autoscaling.k8s.io/verticalpodautoscalers
    - autoscaling/horizontalpodautoscalers
    - batch/cronjobs
    - batch/jobs
    - configmaps
    - discovery.k8s.io/endpointslices
    - endpoints
    - events
    - events.k8s.io/events
    - extensions/daemonsets
    - extensions/deployments
    - extensions/ingresses
    - extensions/replicasets
    - extensions/replicationcontrollers
    - limitranges
    - metrics.k8s.io/nodes
    - metrics.k8s.io/pods
    - namespaces
    - networking.k8s.io/ingresses
    - networking.k8s.io/networkpolicies
    - nodes
    - persistentvolumeclaims
    - persistentvolumes
    - pods
    - pods/log
    - policy/poddisruptionbudgets
    - rbac.authorization.k8s.io/rolebindings
    - rbac.authorization.k8s.io/roles
    - replicationcontrollers
    - resourcequotas
    - serviceaccounts
    - services
    - storage.k8s.io/storageclasses
```

Role `PrivilegedUser` (includes all rules from the role `User`):

```
create:
    - pods/eviction
create,get:
    - pods/attach
    - pods/exec
delete,deletecollection:
    - pods
read:
    - secrets
```

Role `Editor` (includes all rules from the role `User`, `PrivilegedUser`):

```
read-write:
    - apps/deployments
    - apps/statefulsets
    - autoscaling.k8s.io/verticalpodautoscalers
    - autoscaling/horizontalpodautoscalers
    - batch/cronjobs
    - batch/jobs
    - configmaps
    - discovery.k8s.io/endpointslices
    - endpoints
    - extensions/deployments
    - extensions/ingresses
    - networking.k8s.io/ingresses
    - persistentvolumeclaims
    - policy/poddisruptionbudgets
    - serviceaccounts
    - services
write:
    - secrets
```

Role `Admin` (includes all rules from the role `User`, `PrivilegedUser`, `Editor`):

```
create,patch,update:
    - pods
delete,deletecollection:
    - apps/replicasets
    - extensions/replicasets
read-write:
    - deckhouse.io/authorizationrules
    - rbac.authorization.k8s.io/rolebindings
    - rbac.authorization.k8s.io/roles
```

Role `ClusterEditor` (includes all rules from the role `User`, `PrivilegedUser`, `Editor`):

```
read:
    - rbac.authorization.k8s.io/clusterrolebindings
    - rbac.authorization.k8s.io/clusterroles
write:
    - apiextensions.k8s.io/customresourcedefinitions
    - apps/daemonsets
    - extensions/daemonsets
    - storage.k8s.io/storageclasses
```

Role `ClusterAdmin` (includes all rules from the role `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`):

```
read-write:
    - deckhouse.io/clusterauthorizationrules
write:
    - limitranges
    - namespaces
    - networking.k8s.io/networkpolicies
    - rbac.authorization.k8s.io/clusterrolebindings
    - rbac.authorization.k8s.io/clusterroles
    - rbac.authorization.k8s.io/rolebindings
    - rbac.authorization.k8s.io/roles
    - resourcequotas
```

To get an additional list of access rules for a cluster module role (existing user roles and non-standard rules from other Deckhouse modules), use the following command:

```
D8_ROLE_NAME=Editor
d8 k get clusterrole -A -o jsonpath="{range .items[?(@.metadata.annotations.user-
authz\.deckhouse\.io/access-level=='$D8_ROLE_NAME')]}{.rules}{'\n'}{end}" | jq -s add
```

AuthorizationRule example

Use AuthorizationRule to set access rules for users within a specific namespace.

```
apiVersion: deckhouse.io/v1alpha1
kind: AuthorizationRule
metadata:
  name: test-rule
spec:
  accessLevel: Admin
  subjects:
  - kind: Admin
    name: admin@example.com
```

ClusterAuthorizationRule example

Use ClusterAuthorizationRule to set access rules for users either cluster-wide or for specific namespaces.

```
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: test-rule
spec:
  subjects:
  - kind: User
    name: some@example.com
  - kind: ServiceAccount
    name: gitlab-runner-deploy
    namespace: d8-service-accounts
  - kind: Group
    name: some-group-name
  accessLevel: PrivilegedUser
  portForwarding: true
  # Available only with enableMultiTenancy mode turned on (in Enterprise Edition).
  allowAccessToSystemNamespaces: false
  # Available only with enableMultiTenancy mode turned on (in Enterprise Edition).
  namespaceSelector:
    labelSelector:
      matchExpressions:
      - key: stage
        operator: In
        values:
        - test
        - review
      matchLabels:
        team: frontend
```

Extending permissions for high-level roles

To add permissions to a specific high-level role, create a ClusterRole with the annotation `user-authz.deckhouse.io/access-level: <AccessLevel>`.

Example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    user-authz.deckhouse.io/access-level: Editor
  name: user-editor
rules:
- apiGroups:
  - kuma.io
  resources:
  - trafficroutes
  - trafficroutes/finalizers
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
- apiGroups:
  - flagger.app
  resources:
  - canaries
  - canaries/status
  - metrictemplates
  - metrictemplates/status
  - alertproviders
  - alertproviders/status
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
```

Additional resources

- Module `user-authz` documentation

Experimental authorization model

The experimental role model is designed on the principle of aggregation. It combines low-level roles into bigger roles that cover common tasks. This simplifies model extensibility by allowing you to add your own roles.

To use the experimental role model, the `user-authz` module must be enabled in the cluster. This module creates a set of cluster roles (ClusterRole) suitable for most user and group access management tasks.

> ⚠ Starting from Deckhouse Kubernetes Platform (DKP) v1.64, the module includes an experimental role-based access model. The current role model will continue to function, but it will be deprecated in the future.
>
> The current and experimental role-based access models are incompatible. Automatic conversion of resources is not possible.

Unlike the current role model in DKP, the experimental model does not use ClusterAuthorizationRule or AuthorizationRule resources. Access control is configured using the standard Kubernetes RBAC approach via RoleBinding or ClusterRoleBinding resources referencing a role created by the `user-authz` module.

The module creates special aggregated cluster roles (ClusterRole). By using these roles in RoleBinding or ClusterRoleBinding, you can achieve the following:

- Control access to modules associated with a specific DKP subsystem.

  For example, to allow a user acting as a network administrator to configure *network* modules (such as `cni-cilium`, `ingress-nginx`, `istio`, etc.), you can use the `d8:manage:networking:manager` role in a ClusterRoleBinding.

- Control access to *user* module resources within a namespace.

  For example, assigning the `d8:use:role:manager` role via RoleBinding allows users to create, edit, or delete the PodLoggingConfig resource within a namespace. However, it does not grant access to cluster-wide resources such as ClusterLoggingConfig or ClusterLogDestination in the `log-shipper` module, nor does it allow configuring the `log-shipper` module itself.

The roles created by the module fall into two categories:

- Use roles — for granting permissions to users (for example, application developers) *within a specific namespace*.
- Manage roles — for granting administrator permissions.

⚠ Pay attention to the specifics of configuring combined access and shared use of RoleBinding and ClusterAuthorizationRule (CAR) for the same user when multitenancy mode is enabled in the cluster (parameter `enableMultiTenancy: true`). For more details, see the `user-authz` module documentation.

Use roles

⚠ Use rules can only be used in RoleBinding resources.

Use roles are intended for granting permissions to users *within a specific namespace*. “Users” in this context typically refers to developers deploying applications in a cluster configured by an administrator. These users don’t need to manage DKP modules or the cluster, but they do need permissions to create Ingress resources, configure application authentication, collect logs, etc.

Use roles define access to namespaced module resources and standard Kubernetes namespaced resources such as Pod, Deployment, Secret, ConfigMap, etc.

The `user-authz` module creates the following use roles:

Role	Allowed actions	Access restrictions
`d8:use:role:viewer`	View Pod, Deployment, Service (except for secrets and RBAC)	Can’t access `exec`, ports or resource changes
`d8:use:role:user`	Read secrets, `kubectl exec`, `port-forward`, delete Pods, scale replicas	Can’t create or edit objects
`d8:use:role:manager`	Create or modify Pod, ConfigMap, Deployment, manage module resources (such as Certificate)	Can’t access Quota or RBAC
`d8:use:role:admin`	Manage ServiceAccount, Role, ResourceQuota, NetworkPolicy	Full access within a namespace

Key role differences:

- `viewer`: `read-only` (no access to secrets or RBAC).
- `user`: Adds access to secrets, Pods, and networking features (port-forward).
- `manager`: Allows managing application resources and related module resources.
- `admin`: Full control over a namespace (including RBAC and quotas).

Manage roles

⚠ Manage roles do not grant access to namespaces used by user applications.

A manage role only grants access to system namespaces (those starting with `d8-` or `kube-`), and only to those where the modules associated with the role’s subsystem are running.

Manage roles are intended for granting permissions to administer the entire DKP or its parts (specific subsystems), but not for managing user applications. For example, you can use a manage role to allow a security administrator to manage modules responsible for the cluster’s security functions. Such an administrator will be able to manage authentication, authorization, and security policies, but will not have access to other areas of the cluster (such as networking or monitoring subsystems) and will not be able to change anything in user application namespaces.

A manage role grants access to:

- Kubernetes cluster-wide resources.
- Management of DKP modules (ModuleConfig resources) within the role’s subsystem or across all DKP modules if using the `d8:manage:all:*` role.
- Management of DKP module cluster-wide resources within the role’s subsystem or all DKP module resources if using the `d8:manage:all:*` role.
- System namespaces (starting with `d8-` or `kube-`) in which the modules from the role’s subsystem are running or all system namespaces in the case of the `d8:manage:all:*` role.

The naming format for a manage role is `d8:manage:<SUBSYSTEM>:<ACCESS_LEVEL>`, where:

- `SUBSYSTEM` is the role’s subsystem. It can either be one of the listed subsystems, or `all` to cover all subsystems.
- `ACCESS_LEVEL` is the access level.

Examples of manage roles:

- `d8:manage:all:viewer`: View access to the configuration of all DKP modules (ModuleConfig resources), their cluster-wide and namespaced resources, and standard Kubernetes objects (excluding secrets and RBAC resources) in all system namespaces (starting with `d8-` or `kube-`).
- `d8:manage:all:manager`: Same as `d8:manage:all:viewer`, but with `admin`-level access, meaning, permission to view, create,edit, and delete configurations of all DKP modules (ModuleConfig resources), their cluster-wide and namespaced resources, and standard Kubernetes objects in all system namespaces (starting with `d8-` or `kube-`).
- `d8:manage:observability:viewer`: View access to DKP module configurations (ModuleConfig resources) in the `observability` subsystem, their cluster-wide and namespaced resources, and standard Kubernetes objects

(excluding secrets and RBAC resources) in system namespaces such as `d8-log-shipper`, `d8-monitoring`, `d8-okmeter`, `d8-operator-prometheus`, `d8-upmeter`, and `kube-prometheus-pushgateway`.

The module provides two access levels for administrators:

- `viewer`: Allows viewing standard Kubernetes resources, module configurations (ModuleConfig resources), module cluster-wide resources, and namespaced module resources within the module's namespace.
- `manager`: In addition to `viewer` permissions, allows managing standard Kubernetes resources, module configurations (ModuleConfig resources), module cluster-wide resources, and namespaced module resources within the module's namespace.

Role model subsystems

Each DKP module belongs to a specific subsystem. For each subsystem, there is a set of roles with different access levels. These roles are automatically updated when a module is enabled or disabled.

For example, the `networking` subsystem includes the following manage roles, which can be used in ClusterRoleBinding:

- `d8:manage:networking:viewer`
- `d8:manage:networking:manager`

The scope of a role depends on which subsystem it belongs to:

- The scope of roles from the `all` subsystem is all system namespaces (starting with `d8-` or `kube-`) in the cluster.
- The scope of roles from other subsystems includes the namespaces in which the subsystem's modules operate (see the subsystem composition table), as well as all cluster-wide objects of the subsystem's modules.

Role model subsystem composition

Subsystem	Subsystem modules	Namespaces in which modules of a scope operate
all	All modules	All namespaces
deckhouse	• deckhouse • multitenancy-manager	• d8-multitenancy-manager • d8-system
infrastructure	• chrony • cloud-provider-aws • cloud-provider-azure • cloud-provider-dynamix • cloud-provider-gcp • cloud-provider-huaweicloud • cloud-provider-openstack • cloud-provider-vcd • cloud-provider-vsphere • cloud-provider-yandex • cloud-provider-zvirt • descheduler • istio • keepalived • local-path-provisioner • loki • node-manager • terraform-manager • vertical-pod-autoscaler	• d8-chrony • d8-cloud-instance-manager • d8-cloud-provider-aws • d8-cloud-provider-azure • d8-cloud-provider-dynamix • d8-cloud-provider-gcp • d8-cloud-provider-huaweicloud • d8-cloud-provider-openstack • d8-cloud-provider-vcd • d8-cloud-provider-vsphere • d8-cloud-provider-yandex • d8-cloud-provider-zvirt • d8-descheduler • d8-istio • d8-keepalived • d8-local-path-provisioner • d8-monitoring • d8-system • kube-system
kubernetes	• control-plane-manager • flow-schema • kube-dns • kube-proxy • priority-class	• d8-system • kube-system
networking	• cilium-hubble • cni-cilium • cni-flannel • cni-simple-bridge • flow-schema • ingress-nginx • istio • kube-dns • kube-proxy • metallb • network-gateway • network-policy-engine • node-local-dns • openvpn • static-routing-manager	• d8-cni-cilium • d8-cni-flannel • d8-cni-simple-bridge • d8-ingress-nginx • d8-istio • d8-metallb • d8-network-gateway • d8-openvpn • d8-static-routing-manager • d8-system • kube-system

Subsystem	Subsystem modules	Namespaces in which modules of a scope operate
observability	• dashboard • extended-monitoring • log-shipper • monitoring-applications • monitoring-custom • monitoring-deckhouse • monitoring-kubernetes • monitoring-kubernetes-control-plane • monitoring-ping • okmeter • operator-prometheus • prometheus • prometheus-metrics-adapter • prometheus-pushgateway • upmeter	• d8-log-shipper • d8-monitoring • d8-okmeter • d8-operator-prometheus • d8-upmeter • kube-prometheus-pushgateway
security	• admission-policy-engine • cert-manager • operator-trivy • runtime-audit-engine • secret-copier • user-authn • user-authz	• d8-admission-policy-engine • d8-cert-manager • d8-operator-trivy • d8-runtime-audit-engine • d8-user-authn

Creating a new subsystem role

If the current subsystems don’t meet requirements of the role distribution used in the company, you can create a new subsystem that will include roles from the `deckhouse` and `kubernetes` subsystems and the `user-authn` module.

To create a role, use the following template:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: custom:manage:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: admin
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/level: subsystem
    rbac.deckhouse.io/subsystem: custom
    rbac.deckhouse.io/aggregate-to-all-as: manager
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-kubernetes-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        module: user-authn
rules: []
```

Label descriptions

A created role includes the following labels:

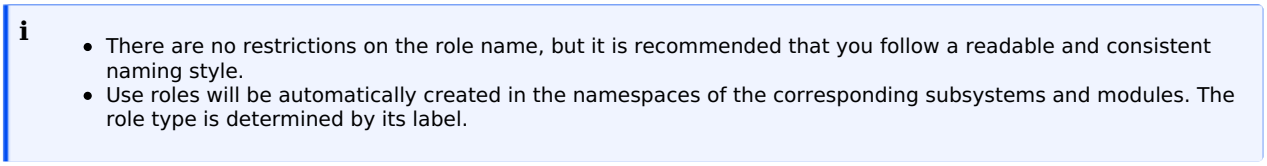
- `rbac.deckhouse.io/use-role: admin`: Indicates which role the hook should use when creating use roles.
- `rbac.deckhouse.io/kind: manage`: Specifies that the role’s type is `manage`. **This label is required**.
- `rbac.deckhouse.io/level: subsystem`: Indicates that the role belongs to a subsystem level and will be processed accordingly.
- `rbac.deckhouse.io/subsystem: custom`: Defines the name of the subsystem this role is responsible for.
- `rbac.deckhouse.io/aggregate-to-all-as: manager`: Allows the `manage:all` role to aggregate this role as a `manager`.

Aggregation selector descriptions

The `aggregationRule` section defines which roles and modules are aggregated into this role:

- `rbac.deckhouse.io/kind: manage`, `rbac.deckhouse.io/aggregate-to-deckhouse-as: manager`: Aggregates a manage role from the `deckhouse` and `kubernetes` subsystems.
- `rbac.deckhouse.io/kind: manage`, `module: user-authn`: Aggregates all rules from the `user-authn` module.

This way, the role inherits permissions from the `deckhouse` and `kubernetes` subsystems as well as from the `user-authn` module.



- There are no restrictions on the role name, but it is recommended that you follow a readable and consistent naming style.
- Use roles will be automatically created in the namespaces of the corresponding subsystems and modules. The role type is determined by its label.

Extending a custom role

In the following example, a new cluster-wide CRD object called MySuperResource has been introduced into the cluster. To grant access to it, you need to extend an existing manage role described above.

1. Add a new aggregation selector to the role:

   ```
   rbac.deckhouse.io/kind: manage
   rbac.deckhouse.io/aggregate-to-custom-as: manager
   ```

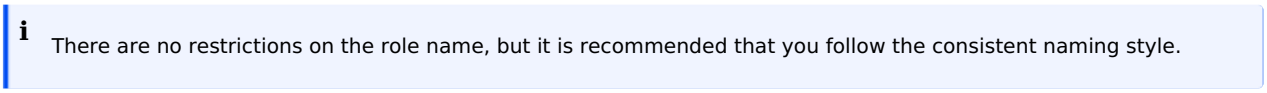
   This selector allows including the role in a subsystem's aggregating role via the corresponding label. After adding the new selector, the role will look as follows:

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRole
   metadata:
     name: custom:manage:mycustom:manager
     labels:
       rbac.deckhouse.io/use-role: admin
       rbac.deckhouse.io/kind: manage
       rbac.deckhouse.io/level: subsystem
       rbac.deckhouse.io/subsystem: custom
       rbac.deckhouse.io/aggregate-to-all-as: manager
   aggregationRule:
     clusterRoleSelectors:
       - matchLabels:
           rbac.deckhouse.io/kind: manage
           rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
       - matchLabels:
           rbac.deckhouse.io/kind: manage
           rbac.deckhouse.io/aggregate-to-kubernetes-as: manager
       - matchLabels:
           rbac.deckhouse.io/kind: manage
           module: user-authn
       - matchLabels:
           rbac.deckhouse.io/kind: manage
           rbac.deckhouse.io/aggregate-to-custom-as: manager
   rules: []
   ```

2. Create a new role to define permissions for the new resource. Example configuration for the read-only permissions:

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRole
   metadata:
     labels:
       rbac.deckhouse.io/aggregate-to-custom-as: manager
       rbac.deckhouse.io/kind: manage
     name: custom:manage:permission:mycustom:superresource:view
   rules:
   - apiGroups:
     - mygroup.io
     resources:
     - mysuperresources
     verbs:
     - get
     - list
     - watch
   ```

   The new role's permissions will be added to the subsystem role, providing view access to the new object.



There are no restrictions on the role name, but it is recommended that you follow the consistent naming style.

Extending existing manage-subsystem roles

To extend an existing role, follow the same steps as above but make sure to change the role name and labels.

Example configuration for extending the manage role from the `deckhouse` subsystem (`d8:manage:deckhouse:manager`):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    rbac.deckhouse.io/kind: manage
  name: custom:manage:permission:mycustommodule:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

This way the new role will extend the `d8:manage:deckhouse` role.

Extending manage-subsystem roles and adding a namespace

To add a new namespace for creating a use role in it via the hook, you would need only one label:

```
"rbac.deckhouse.io/namespace": namespace
```

This label informs the hook that a use role needs to be created in this namespace:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/namespace: namespace
  name: custom:manage:permission:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

The hook watches ClusterRoleBinding resources and, upon their creation, inspects the associated manage roles to find all aggregated roles defined via the `aggregationRule`. For each aggregated role, it extracts the namespace from the `rbac.deckhouse.io/namespace` label and creates a corresponding use role in that namespace.

Extending existing use roles

If a resource is part of a namespace, you will need to extend the use role instead of the manage role. The only difference is the name and label:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-kubernetes-as: user
    rbac.deckhouse.io/kind: use
  name: custom:use:capability:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

This role extends the `d8:use:role:user:kubernetes` role.

Additional resources

- Module `cni-cilium` documentation
- Module `ingress-nginx` documentation
- Module `istio` documentation
- Module `log-shipper` custom resources
- Module `user-authn` documentation
- Module `user-authz` documentation

Cluster admin access model

Deckhouse Kubernetes Platform (DKP) supports the presence of multiple kubeconfig files on master nodes (this feature is implemented by the `control-plane-manager` module). Understanding their purpose is important for secure cluster administration.

Kubeconfig files on master nodes

The following kubeconfig files are located on the master nodes:

File	Identity	Purpose
`/etc/kubernetes/admin.conf`	`kubernetes-admin` (`kubeadm:cluster-admins` group)	Machine kubeconfig for kubeadm internals (join, renewal). With the `user-authz` module enabled, RBAC uses `user-authz:cluster-admin` plus an additional ClusterRole. With `user-authz` disabled, the group is bound to the built-in `cluster-admin` role
`/etc/kubernetes/super-admin.conf`	`kubernetes-super-admin` (`system:masters` group)	Break-glass emergency credential. Bypasses RBAC entirely. Restrict access to this file to trusted recovery scenarios
`/etc/kubernetes/controller-manager.conf`	`system:kube-controller-manager`	Used by kube-controller-manager
`/etc/kubernetes/scheduler.conf`	`system:kube-scheduler`	Used by kube-scheduler

RBAC-based admin access

Starting from Kubernetes 1.29, kubeadm generates `admin.conf` with the `kubeadm:cluster-admins` group instead of `system:masters`. This provides RBAC-controlled admin access that can be revoked by removing the `kubeadm:cluster-admins` ClusterRoleBinding(s).

If the `user-authz` module is **disabled**, DKP binds the `kubeadm:cluster-admins` group to the built-in wildcard ClusterRole `cluster-admin` (same effective model as a plain kubeadm cluster without extra RBAC).

If `user-authz` is **enabled**, the group is bound to `user-authz:cluster-admin`, and a second ClusterRoleBinding adds ClusterRole `d8:control-plane-manager:admin-kubeconfig-supplement` (rules beyond the high-level role, e.g. for certificates and cluster machinery). Together they replace a single wildcard `cluster-admin` for this identity. For full unrestricted access, use `super-admin.conf`.

Recommended admin access

If the `user-authn` module is enabled, use personalized OIDC-based kubeconfig obtained through the kubeconfig generator. This provides individual accountability and audit trail.

If `user-authn` is disabled, administrators can explicitly use the admin kubeconfig on a master node:

```
d8 k --kubeconfig=/etc/kubernetes/admin.conf <command>
```

Root kubeconfig symlink

By default, the `control-plane-manager` module creates a symlink `/root/.kube/config` → `/etc/kubernetes/admin.conf` on master nodes, allowing root to run `d8 k` without specifying `--kubeconfig`.

When the `user-authz` module is enabled, you can disable this symlink by setting `rootKubeconfigSymlink: false` in the `control-plane-manager` module configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    rootKubeconfigSymlink: false
```

If `user-authz` is disabled, `control-plane-manager` does not apply this setting and keeps the default symlink behavior.

When the symlink is disabled (and `user-authz` is enabled), the symlink is removed when it pointed to `admin.conf`. Use personalized credentials or pass `--kubeconfig` explicitly.

Security hardening

The `control-plane-manager` module automatically restricts file permissions on `admin.conf` and `super-admin.conf` to `0600` (owner read/write only) during every reconciliation cycle. This prevents unauthorized users from reading these sensitive credentials.

Break-glass access

In emergency situations (RBAC misconfiguration, webhook failures), use `super-admin.conf`:

```
d8 k --kubeconfig=/etc/kubernetes/super-admin.conf <command>
```

This credential bypasses all RBAC checks. Use it only as a last resort and restrict who can read the file.

Additional resources

- Module `control-plane-manager` documentation
- Module `user-authn` documentation
- Module `user-authz` documentation

Granting permissions to users and service accounts

To grant permissions in Deckhouse Kubernetes Platform (DKP), you need to define a `subjects` block in custom resources.

For users, it should be specified in the following format:

```
subjects:
- kind: User
  name: <user email>
```

⚠ If you are using the `user-authn` module and static users, make sure to specify the user's email in `subjects`, not the name of the User resource.

Alternatively, you can grant permissions by group:

```
subjects:
- kind: Group
  name: <name of the group the user belongs to>
```

For a service account, the `subjects` block should be specified as follows:

```
subjects:
- kind: ServiceAccount
  name: <service account name>
  namespace: <namespace where the service account is created>
```

Granting permissions using AuthorizationRule and ClusterAuthorizationRule (current role model)

When using the current role model in DKP, you can grant permissions to users via the AuthorizationRule and ClusterAuthorizationRule resources.

Granting permissions to a user within a single namespace

To grant a user permissions within a single namespace, use the AuthorizationRule resource. It is applied within a single namespace.

Example:

```
apiVersion: deckhouse.io/v1
kind: AuthorizationRule
metadata:
  name: dev-access
  namespace: dev-namespace
spec:
  subjects:
  - kind: User
    name: dev-user@example.com
  accessLevel: Admin
  portForwarding: true
```

Granting permissions to a user in all namespaces

To grant a user permissions across all namespaces, including system ones (for example, to assign administrator permissions), use the ClusterAuthorizationRule resource. It is applied cluster-wide.

If needed, you can restrict the scope of permissions granted via ClusterAuthorizationRule to one or several namespaces. To do this, you can set the corresponding restrictions in the manifest. However, if possible, it is recommended that you use the AuthorizationRule resource for this purpose.

Example:

```
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: admin-access
spec:
  subjects:
  - kind: User
    name: dev-user@example.com
  # Available only with enableMultiTenancy mode turned on
  # in the user-authz module (Enterprise Edition).
  namespaceSelector:
    labelSelector:
      matchLabels:
        env: review
  accessLevel: SuperAdmin
  portForwarding: true
```

Granting permissions using ClusterRoleBinding and RoleBinding (experimental role model)

When using the experimental role model in DKP, you can grant permissions to users via the ClusterRoleBinding and RoleBinding resources.

Assigning cluster administrator permissions (experimental role model)

To assign cluster administrator permissions, use the manage role `d8:manage:all:manager` in a ClusterRoleBinding resource.

Example of assigning cluster administrator permissions to the user `jane`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin-jane
subjects:
- kind: User
  name: jane.doe@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:manage:all:manager
  apiGroup: rbac.authorization.k8s.io
```

Permissions the user will obtain

The user's permissions will be limited to namespaces starting with `d8-` or `kube-`.

The user will have the following permissions:

- View, modify, delete, and create Kubernetes resources and DKP module resources.
- Modify module configurations (view, edit, delete, and create ModuleConfig resources).
- Run the following commands on pods and services:
  - `kubectl attach`
  - `kubectl exec`
  - `kubectl port-forward`
  - `kubectl proxy`

Assigning networking administrator permissions (experimental role model)

To assign networking administrator permissions for managing the cluster's networking subsystem, use the manage role `d8:manage:networking:manager` in a ClusterRoleBinding resource.

Example of assigning networking administrator permissions to the user `jane`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: network-admin-jane
subjects:
- kind: User
  name: jane.doe@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:manage:networking:manager
  apiGroup: rbac.authorization.k8s.io
```

Permissions the user will obtain

The user's permissions will be limited to the following DKP module namespaces from the networking subsystem (the actual list depends on the modules enabled in the cluster):

- `d8-cni-cilium`
- `d8-cni-flannel`
- `d8-cni-simple-bridge`
- `d8-ingress-nginx`
- `d8-istio`
- `d8-metallb`
- `d8-network-gateway`
- `d8-openvpn`
- `d8-static-routing-manager`
- `d8-system`
- `kube-system`

The user will have the following permissions:

- View, modify, create, and delete *standard* Kubernetes resources in module namespaces from the `networking` subsystem.

  Example resources the user will be able to manage (not a complete list):

  - Certificate
  - CertificateRequest
  - ConfigMap
  - ControllerRevision
  - CronJob
  - DaemonSet
  - Deployment
  - Event
  - HorizontalPodAutoscaler
  - Ingress
  - Issuer
  - Job
  - Lease
  - LimitRange
  - NetworkPolicy
  - PersistentVolumeClaim
  - Pod
  - PodDisruptionBudget
  - ReplicaSet
  - ReplicationController
  - ResourceQuota
  - Role
  - RoleBinding
  - Secret
  - Service
  - ServiceAccount

    - StatefulSet
    - VerticalPodAutoscaler
    - VolumeSnapshot

- View, modify, create, and delete resources in DKP module namespaces from the `networking` subsystem.

  Resources the user will be able to manage:

    - EgressGateway
    - EgressGatewayPolicy
    - FlowSchema
    - IngressClass
    - IngressIstioController
    - IngressNginxController
    - IPRuleSet
    - IstioFederation
    - IstioMulticluster
    - RoutingTable

- Modify module configurations (view, modify, create, and delete ModuleConfig resources) in the `networking` subsystem.

  Modules the user will be able to manage:

    - `cilium-hubble`
    - `cni-cilium`
    - `cni-flannel`
    - `cni-simple-bridge`
    - `flow-schema`
    - `ingress-nginx`
    - `istio`
    - `kube-dns`
    - `kube-proxy`
    - `metallb`
    - `network-gateway`
    - `network-policy-engine`
    - `node-local-dns`
    - `openvpn`
    - `static-routing-manager`

- Run the following commands on Pods and Services in the module namespaces within the `networking` subsystem:

    - `kubectl attach`
    - `kubectl exec`
    - `kubectl port-forward`
    - `kubectl proxy`

Granting administrator permissions to a user within a namespace (experimental role model)

To assign or restrict user permissions to specific namespaces, apply a use role with the corresponding access level in a RoleBinding resource.

For example, to allow a user to manage application resources in a namespace (without giving them access to DKP module configurations), use the `d8:use:role:admin` role in a RoleBinding resource for the corresponding namespace.

Example of granting application developer `app-developer` permissions within the `myapp` namespace:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: myapp-developer
  namespace: myapp
subjects:
- kind: User
  name: app-developer@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:use:role:admin
  apiGroup: rbac.authorization.k8s.io
```

Permissions the user will obtain

The user’s permissions will be limited to the following within the `myapp` namespace:

- View, modify, create, and delete Kubernetes resources. Example resources:

    - Certificate
    - CertificateRequest
    - ConfigMap
    - ControllerRevision
    - CronJob
    - DaemonSet
    - Deployment

    - Event
    - HorizontalPodAutoscaler
    - Ingress
    - Issuer
    - Job
    - Lease
    - LimitRange
    - NetworkPolicy
    - PersistentVolumeClaim
    - Pod
    - PodDisruptionBudget
    - ReplicaSet
    - ReplicationController
    - ResourceQuota
    - Role
    - RoleBinding
    - Secret
    - Service
    - ServiceAccount
    - StatefulSet
    - VerticalPodAutoscaler
    - VolumeSnapshot
- View, modify, create, and delete the following DKP module resources:
    - DexAuthenticator
    - DexClient
    - PodLoggingConfig
- Run the following commands on Pods and Services:
    - `kubectl attach`
    - `kubectl exec`
    - `kubectl port-forward`
    - `kubectl proxy`

Additional resources

- Module `user-authn` documentation
- Module `user-authz` custom resources

Checking user permissions

To check whether a user has the necessary permissions, run the following command, which includes:

- `resourceAttributes` (as in RBAC): Permission checking target.
- `user`: User name.
- `groups`: User groups.

> **i** If you're using the `user-authn` module, you can see the user's name and groups in the Dex logs (only logged during authorization) by running:
>
> ```
> d8 k -n d8-user-authn logs -l app=dex
> ```

```
cat  <<EOF | 2>&1 d8 k create --raw  /apis/authorization.k8s.io/v1/subjectaccessreviews -f - | jq
.status
{
  "apiVersion": "authorization.k8s.io/v1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "",
      "verb": "watch",
      "version": "v1",
      "resource": "pods"
    },
    "user": "system:kube-controller-manager",
    "groups": [
      "Admins"
    ]
  }
}
EOF
```

The response will show whether access is allowed and which role grants it.

Example response if the user has access permissions:

```
{
  "allowed": true,
  "reason": "RBAC: allowed by ClusterRoleBinding \"system:kube-controller-manager\" of ClusterRole
\"system:kube-controller-manager\" to User \"system:kube-controller-manager\""
}
```

Example response if the user does not have access permissions:

```
{
  "allowed": false
}
```

If **multitenancy** mode is enabled in the cluster, run an additional check to verify that the user has access to the namespace:

```
cat  <<EOF | 2>&1 d8 k --kubeconfig /etc/kubernetes/deckhouse/extra-files/webhook-config.yaml create
--raw / -f - | jq .status
{
  "apiVersion": "authorization.k8s.io/v1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "",
      "verb": "watch",
      "version": "v1",
      "resource": "pods"
    },
    "user": "system:kube-controller-manager",
    "groups": [
      "Admins"
    ]
  }
}
EOF
```

Example response if the user has access permissions:

```
{
  "allowed": false
}
```

A response with `"allowed": false` means the webhook is not blocking the request. If the webhook does block the request, you will see an error message like this:

```
{
  "allowed": false,
  "denied": true,
  "reason": "making cluster scoped requests for namespaced resources are not allowed"
}
```

Additional resources

- Module `user-authn` documentation

Access for CI/CD

The following methods are available for authenticating CI/CD pipelines in the Kubernetes API:

- ServiceAccount — Kubernetes ServiceAccount token.
- Basic Auth — username and password via IdP.
- Token Exchange — exchange IdP token for Dex token.

ServiceAccount

The ServiceAccount token is used directly for API authentication. No external IdP is required. For ServiceAccount, you can create long-lived tokens (without TTL, with the possibility of revocation or rotation) and short-lived tokens.

When multiple pipelines share a single ServiceAccount, audit logs will not contain information about specific pipelines.

Prerequisites

To configure token-based authentication for ServiceAccount, the following requirements must be met:

- Cluster access with permissions to create ServiceAccounts and Secrets.

- For external access: publishAPI or direct API access via VPN.

Create ServiceAccount and a long-lived token

To create a ServiceAccount and token, use the command:

```
d8 k create ns ci-deploy || true

cat <<EOF | d8 k apply -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gitlab-runner-deploy
  namespace: ci-deploy
---
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-runner-deploy-token
  namespace: ci-deploy
  annotations:
    kubernetes.io/service-account.name: gitlab-runner-deploy
type: kubernetes.io/service-account-token
EOF
```

> **i** The `kubernetes.io/service-account-token` Secret type is a legacy approach. The recommended method is the TokenRequest API (`d8 k create token ...`).

Grant permissions

For details on granting permissions, see Granting permissions to users and service accounts.

Assign the necessary permissions to the ServiceAccount.

For the current role model, use ClusterAuthorizationRule:

```
cat <<EOF | d8 k apply -f -
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: gitlab-runner-access
spec:
  subjects:
  - kind: ServiceAccount
    name: gitlab-runner-deploy
    namespace: ci-deploy
  accessLevel: Admin
  portForwarding: true
EOF
```

Available levels: `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, `ClusterAdmin`, `SuperAdmin`.

For the experimental role model, use ClusterRoleBinding:

```
cat <<EOF | d8 k apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: gitlab-runner-access
subjects:
- kind: ServiceAccount
  name: gitlab-runner-deploy
  namespace: ci-deploy
roleRef:
  kind: ClusterRole
  name: d8:manage:all:manager
  apiGroup: rbac.authorization.k8s.io
EOF
```

Get API URL and CA certificate

When using publishAPI:

```
API_HOST=$(d8 k -n d8-user-authn get ingress kubernetes-api -o jsonpath='{.spec.rules[0].host}')
echo "API endpoint: https://${API_HOST}"
```

> **i** If the API certificate is signed by a public CA (Let's Encrypt), the `--certificate-authority` parameter is not required.

For private CA:

```
d8 k -n d8-user-authn get secret kubernetes-api-ca-key-pair -o jsonpath='{.data.ca\.crt}' | base64 -d > /tmp/ca.crt
```

Create kubeconfig

Example of generating kubeconfig for a previously created ServiceAccount:

```
export CLUSTER_NAME=my-cluster
export USER_NAME=gitlab-runner-deploy
export CONTEXT_NAME=${CLUSTER_NAME}-${USER_NAME}
export FILE_NAME=kube.config

# For public CA, remove --certificate-authority and --embed-certs
d8 k config set-cluster $CLUSTER_NAME \
  --server=https://${API_HOST} \
  --certificate-authority=/tmp/ca.crt \
  --embed-certs=true \
  --kubeconfig=$FILE_NAME

d8 k config set-credentials $USER_NAME \
  --token=$(d8 k -n ci-deploy get secret gitlab-runner-deploy-token -o jsonpath='{.data.token}' | base64 -d) \
  --kubeconfig=$FILE_NAME

d8 k config set-context $CONTEXT_NAME \
  --cluster=$CLUSTER_NAME --user=$USER_NAME \
  --kubeconfig=$FILE_NAME

d8 k config use-context $CONTEXT_NAME --kubeconfig=$FILE_NAME
```

Short-lived tokens (TokenRequest)

The TokenRequest API allows creating tokens with limited lifetime. When creating such a token, specify the name of the ServiceAccount that needs to be granted access in the command (in the example — `gitlab-runner-deploy`):

```
d8 k create token gitlab-runner-deploy -n ci-deploy --duration=1h
```

> **⚠** The `d8 k create token` command itself requires access to the Kubernetes API. If the CI job has no initial cluster access, it cannot execute TokenRequest from within the job.

Usage patterns in CI

Bootstrap token with minimal permissions. CI stores a long-lived kubeconfig/token that only has permission to create TokenRequest for a specific ServiceAccount. At the start of the job, the bootstrap token issues a short-lived token used for deployment.

External token broker. An operator, secret manager (Vault, External Secrets), or external service periodically issues short-lived tokens and places them in CI variables or secrets. The CI job only consumes the ready token.

> **i** The bootstrap token should have minimal permissions: only `create` on the `serviceaccounts/token` resource for a specific ServiceAccount/namespace. Do not use cluster-admin. Limit the TTL of issued tokens and promptly revoke bootstrap access if compromised.

Usage without kubeconfig file:

```
export KUBE_SERVER="https://${API_HOST}"
export KUBE_TOKEN=$(d8 k create token gitlab-runner-deploy -n ci-deploy --duration=1h)
d8 k --server=$KUBE_SERVER --token=$KUBE_TOKEN get ns
```

Basic Auth

Authentication via username and password through IdP (LDAP, OIDC).

⚠ The password is sent to DKP and validated through basic-auth-proxy/Dex.

⚠ Only one DexProvider in the cluster can have `enableBasicAuth: true`.

Prerequisites

The following requirements must be met to configure Basic Auth:

- publishAPI enabled.
- DexProvider configured for IdP.

Enable

Add `enableBasicAuth: true` to DexProvider. For DexProvider configuration examples, see user-authn module documentation.

Get API endpoint

To get the API endpoint, use the command:

```
API_HOST=$(d8 k -n d8-user-authn get ingress kubernetes-api -o jsonpath='{.spec.rules[0].host}')
echo "https://${API_HOST}"
```

Verification

To check access, use the command:

```
curl -q -u "$K8S_USER:$K8S_PASSWORD" "https://${API_HOST}/version"
```

Possible errors:

- `401` — invalid credentials or Basic Auth not enabled.
- `403` — authentication succeeded but RBAC denies access.

Configure kubeconfig

Example of kubeconfig configuration for Basic Auth:

```
apiVersion: v1
kind: Config
clusters:
- name: my-cluster
  cluster:
    server: https://<API_HOST>
users:
- name: basic-auth-user
  user:
    username: "<USERNAME>"
    password: "<PASSWORD>"
contexts:
- name: default
  context:
    cluster: my-cluster
    user: basic-auth-user
current-context: default
```

Use in GitLab CI

Example of using Basic Auth in GitLab CI:

```
deploy:
  script:
    - d8 k --server="$K8S_SERVER" --username="$K8S_USER" --password="$K8S_PASSWORD" get ns
```

Variables `K8S_SERVER`, `K8S_USER`, `K8S_PASSWORD` are set in the project CI/CD settings.

Token Exchange

The pipeline obtains a token from the IdP, exchanges it at Dex for a token with `aud=kubernetes`, and uses it to access the API.

> **i** Recommended for GitLab CI and GitHub Actions.

DKP/Dex does not receive the user password. How `IDP_TOKEN` is obtained depends on the IdP: OIDC job token (GitLab/GitHub) or IdP token endpoint (client_credentials).

Prerequisites

The following requirements must be met to configure Token Exchange:

- publishAPI enabled.
- DexProvider configured as **OIDC type**.

> **⚠** Token exchange does not work with `type: GitLab` or `type: GitHub` connectors. Configure GitLab/GitHub as OIDC providers instead.

Create DexClient

To create DexClient, use the command:

```
cat <<EOF | d8 k apply -f -
apiVersion: deckhouse.io/v1
kind: DexClient
metadata:
  name: ci-token-exchange
  namespace: d8-user-authn
  annotations:
    dexclient.deckhouse.io/allow-access-to-kubernetes: "true"
spec: {}
EOF
```

The annotation `dexclient.deckhouse.io/allow-access-to-kubernetes` allows the client to request tokens with `aud=kubernetes`.

> **⚠** This grants cluster-wide access to the Kubernetes API. For this reason, adding the annotation or changing it to `"true"` requires access to modify the `user-authn` module configuration — such as the `d8:manage:permission:module:user-authn:edit` role. A namespace-level role that permits creating a DexClient is not enough: the admission-controller rejects the request with a message naming the annotation.
>
> The addition of the annotation is restricted regardless of the specified value, including `"false"`. This is required to ensure compatibility with previous DKP versions where access has been granted when the annotation was present, regardless of its value.
>
> This restriction is not applicable to an object, which already has the annotation assigned. Deletion and recreation of DexClient is treated as another addition of the annotation, so a GitOps controller that recreates the object instead of patching it requires the `update` permission on the `moduleconfigs` resource named `user-authn`.

Get client secret:

```
d8 k -n d8-user-authn get secret dex-client-ci-token-exchange -o jsonpath='{.data.clientSecret}' | base64 -d
```

client_id format: `dex-client-ci-token-exchange@d8-user-authn`.

Get Dex and API URLs

To get the Dex URL and API, use the following commands:

```
DEX_HOST=$(d8 k -n d8-user-authn get ingress dex -o jsonpath='{.spec.rules[0].host}')
API_HOST=$(d8 k -n d8-user-authn get ingress kubernetes-api -o jsonpath='{.spec.rules[0].host}')
```

Grant RBAC

DKP configures kube-apiserver to validate Dex tokens. The `email` and `groups` claims from the token are used for RBAC.

The set of claims required by kube-apiserver for authentication depends on configuration. If kube-apiserver requires the `name` claim, add the `profile` scope.

For details on granting permissions, see Granting permissions to users and service accounts .

For the current role model, use ClusterAuthorizationRule:

```
cat <<EOF | d8 k apply -f -
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: ci-deployer
spec:
  subjects:
  - kind: User
    name: deployer@example.com
  accessLevel: Admin
EOF
```

For the experimental role model, use ClusterRoleBinding:

```
cat <<EOF | d8 k apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: ci-deployer
subjects:
- kind: User
  name: deployer@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:manage:all:manager
  apiGroup: rbac.authorization.k8s.io
EOF
```

Token exchange request

Example of a token exchange request:

```
RESPONSE=$(curl -q -s -X POST "https://${DEX_HOST}/token" \
  -u "${DEX_CLIENT_ID}:${DEX_CLIENT_SECRET}" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
  -d "subject_token=${IDP_TOKEN}" \
  -d "subject_token_type=urn:ietf:params:oauth:token-type:id_token" \
  -d "connector_id=${CONNECTOR_ID}" \
  -d "scope=openid profile email groups audience:server:client_id:kubernetes" \
  -d "requested_token_type=urn:ietf:params:oauth:token-type:id_token")

DEX_TOKEN=$(echo "$RESPONSE" | jq -r '.access_token')
```

Parameters:

- `subject_token` — token from IdP
- `subject_token_type` — `id_token` (GitLab/GitHub) or `access_token` (Keycloak)
- `connector_id` — `metadata.name` of the DexProvider resource
- `scope` — must include `audience:server:client_id:kubernetes` and `profile`

> **i** The expected audience is always `kubernetes`. If the API returns 401 with an audience mismatch error, verify that the scope includes `audience:server:client_id:kubernetes`.

GitLab CI

Example of a token request for GitLab CI:

```
deploy:
  id_tokens:
    GITLAB_OIDC_TOKEN:
      aud: https://<DEX_HOST>/
  script:
    - |
      RESPONSE=$(curl -q -s -X POST "https://${DEX_HOST}/token" \
        -u "${DEX_CLIENT_ID}:${DEX_CLIENT_SECRET}" \
        -H "Content-Type: application/x-www-form-urlencoded" \
        -d "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
        -d "subject_token=${GITLAB_OIDC_TOKEN}" \
        -d "subject_token_type=urn:ietf:params:oauth:token-type:id_token" \
        -d "connector_id=gitlab" \
        -d "scope=openid profile email groups audience:server:client_id:kubernetes" \
        -d "requested_token_type=urn:ietf:params:oauth:token-type:id_token")
      DEX_TOKEN=$(echo "$RESPONSE" | jq -r '.access_token')
      d8 k --server="${K8S_SERVER}" --token="${DEX_TOKEN}" auth whoami
```

Keycloak

Get token from Keycloak:

```
KEYCLOAK_TOKEN=$(curl -q -s -X POST "https://<KEYCLOAK_HOST>/realms/<REALM>/protocol/openid-
connect/token" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=client_credentials" \
  -d "client_id=<KEYCLOAK_CLIENT_ID>" \
  -d "client_secret=<KEYCLOAK_CLIENT_SECRET>" | jq -r '.access_token')
```

Exchange at Dex (for Keycloak, use `subject_token_type=access_token`):

> **i** For access_token exchange, DexProvider requires `getUserInfo: true`.

```
RESPONSE=$(curl -q -s -X POST "https://${DEX_HOST}/token" \
  -u "${DEX_CLIENT_ID}:${DEX_CLIENT_SECRET}" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
  -d "subject_token=${KEYCLOAK_TOKEN}" \
  -d "subject_token_type=urn:ietf:params:oauth:token-type:access_token" \
  -d "connector_id=keycloak" \
  -d "scope=openid profile email groups audience:server:client_id:kubernetes" \
  -d "requested_token_type=urn:ietf:params:oauth:token-type:id_token")

DEX_TOKEN=$(echo "$RESPONSE" | jq -r '.access_token')
d8 k --server="https://${API_HOST}" --token="${DEX_TOKEN}" auth whoami
```

Diagnostics

> **i** Use `d8 k auth whoami` to verify which identity is being used.

`--token` flag may not work — kubeconfig with `auth-provider`, `exec`, or `tokenFile` overrides the token from the flag. Use `-v=8` for debugging:

```
d8 k --server="https://${API_HOST}" --token="${DEX_TOKEN}" auth whoami -v=8
```

The output shows which Authorization header is sent. For reliable token usage, use a minimal kubeconfig with only `server` and `token`, without `auth-provider`/`exec`.

Dex 400 — invalid `subject_token`, `subject_token_type`, or `connector_id`.

Dex 401 — invalid client credentials or invalid subject token.

API 401 — token validation failed. Check:

- Annotation `dexclient.deckhouse.io/allow-access-to-kubernetes` on DexClient. If it is gone, check whether the admission-controller rejected the request that was supposed to add it, or whether a GitOps controller recreated the object without it: only a subject allowed to update the `user-authn` ModuleConfig may add the annotation.
- Scope contains `audience:server:client_id:kubernetes` and `profile`.
- Time synchronization between CI runner and cluster.

API 403 — authentication succeeded but RBAC does not allow access for the user or group from the token.

Decode token to check claims:

```
echo "${DEX_TOKEN}" | cut -d. -f2 | base64 -d 2>/dev/null | jq .
```

Additional resources

- Module `user-authn` documentation
- Module `user-authz` custom resources

Integration with IaaS

Deckhouse Kubernetes Platform (DKP) provides built-in tools for integrating with various cloud providers and virtualization systems. These tools let you:

- Automatically use cloud infrastructure to provision virtual machines and connect them to the cluster.
- Deploy clusters in cloud environments.
- Scale resources as needed.
- Automatically create base infrastructure resources (such as networks, routes, etc.), if supported by the provider.

The workflow for working with different cloud providers is generally the same, with only the preparation steps (such as creating a service account in the cloud) and configuration files differing.

Supported cloud providers:

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure
- OpenStack
- OVH Cloud
- Selectel Cloud
- VK Cloud
- Yandex Cloud

Integration is also possible with private clouds based on the following solutions:

- VK Cloud
- OpenStack
- Huawei Cloud

In addition to cloud providers, integration is supported with the following virtualization systems:

- Deckhouse Virtualization Platform (DVP)
- VMware Cloud Director
- VMware vSphere
- zVirt

It is also possible to configure hybrid clusters.

Public cloud providers

Deckhouse Kubernetes Platform (DKP) can be deployed in various public clouds, enabling the use of scalable infrastructure and flexible integration with cloud services.

DKP supports the following providers:

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure
- OpenStack
- OVH Cloud
- Selectel Cloud
- VK Cloud
- Yandex Cloud

This simplifies the choice of deployment environment and enables a unified approach to managing Kubernetes clusters across different cloud ecosystems.

Connection and authorization in Yandex Cloud

To allow Deckhouse Kubernetes Platform (DKP) to manage resources in Yandex Cloud, you need to:

- Create a service account.
- Assign the required IAM roles to it.
- Generate an authorization key.
- Optionally reserve a public IP address.
- Ensure sufficient resource quotas in the cloud.

The steps below guide you through setting up the connection.

Preparing the environment

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

The `cloud-init` package must be installed on virtual machines (VM). After a VM is started, the following services associated with this package must be running:

- `cloud-config.service`
- `cloud-final.service`
- `cloud-init.service`

To verify that the services are running, use the following commands:

```
systemctl status cloud-config.service
systemctl status cloud-final.service
systemctl status cloud-init.service
```

Creating a service account

To enable DKP to manage Yandex Cloud resources, create a service account and assign it editing permissions. You can find detailed instructions on service account creation in the Yandex Cloud documentation.

To create the service account, run the following command:

```
yc iam service-account create --name deckhouse
```

The command will return information about the newly created service account:

```
id: <userID>
folder_id: <folderID>
created_at: "YYYY-MM-DDTHH:MM:SSZ"
name: deckhouse
```

⚠ Save the `userID` and `folderID` as they will be needed in the following steps.

Assigning IAM roles

To allow DKP to work with cloud resources, assign the following roles to the service account:

```
yc resource-manager folder add-access-binding --id <folderID> --role compute.editor --subject serviceAccount:<userID>
yc resource-manager folder add-access-binding --id <folderID> --role vpc.admin --subject serviceAccount:<userID>
yc resource-manager folder add-access-binding --id <folderID> --role load-balancer.editor --subject serviceAccount:
<userID>
```

Generating an authorization key

Generate a JSON authorization file to use in your configuration:

```
yc iam key create --service-account-name deckhouse --output deckhouse-sa-key.json
```

Use the contents of the `deckhouse-sa-key.json` file in the `provider.serviceAccountJSON` field when defining the cluster configuration.

Checking and increasing quotas

Make sure your cloud account has the required quotas for cluster deployment and scaling:

- vCPUs: 64
- SSD storage: 2000 GB
- Number of VMs: 25
- RAM: 256 GB

Increase quotas via the Yandex Cloud Console if necessary.

Reserving a public IP

If you are using the WithoutNAT or WithNATInstance deployment layout and need a fixed external IP address (for example, to use in `externalIPAddresses`, `natInstanceExternalAddress`, or a bastion host), run the following command:

```
yc vpc address create --external-ipv4 zone=ru-central1-a
```

Example output:

```
id: e9b4cfmmnc1mhgij75n7
folder_id: b1gog0h9k05lhqe5d88l
created_at: "2020-09-01T09:29:33Z"
external_ipv4_address:
  address: 178.154.226.159
  zone_id: ru-central1-a
  requirements: {}
reserved: true
```

After completing these steps, you will have all the necessary information to create a YandexClusterConfiguration resource that describes your cluster in Yandex Cloud.

Additional resources

- Module `cloud-provider-yandex` provider configuration

Layouts and configuration in Yandex Cloud

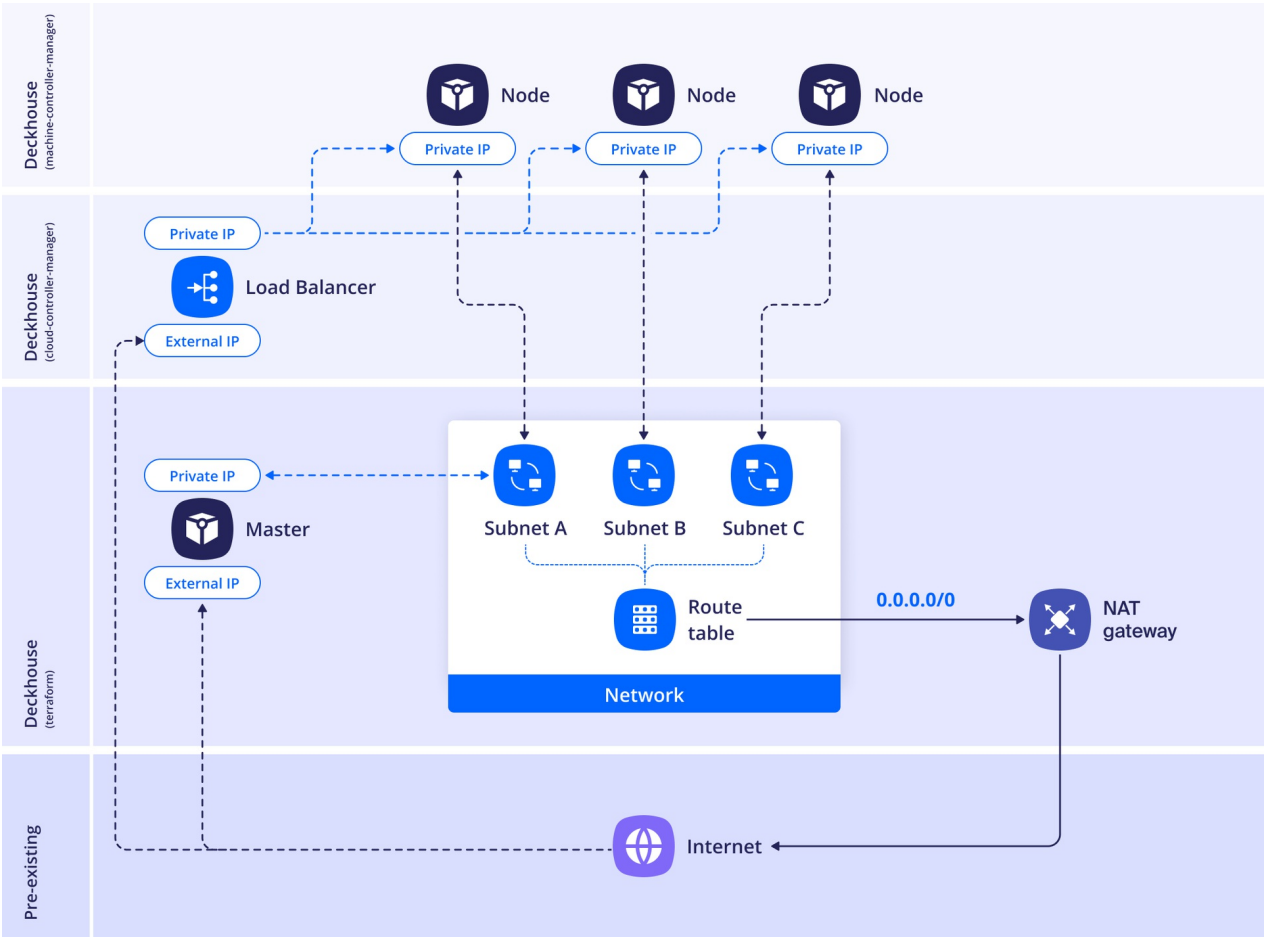
Layouts

This section describes the possible node placement layouts in Yandex Cloud infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

✖ In this layout, nodes will not have public IP addresses and will access the internet via a Yandex Cloud NAT Gateway. The NAT Gateway uses random public IP addresses from dedicated address ranges. Because of this, you cannot whitelist the IP addresses of cloud resources behind a specific NAT Gateway in third-party services.

⚠ Because nodes are created without public IP addresses in this layout, the master node must be accessible over SSH from the machine where `dhctl` is running: either directly over a private network or through a bastion host. If the master node is not directly accessible, run the installation with the `--ssh-bastion-host`, `--ssh-bastion-user`, and, if needed, `--ssh-bastion-port` parameters.



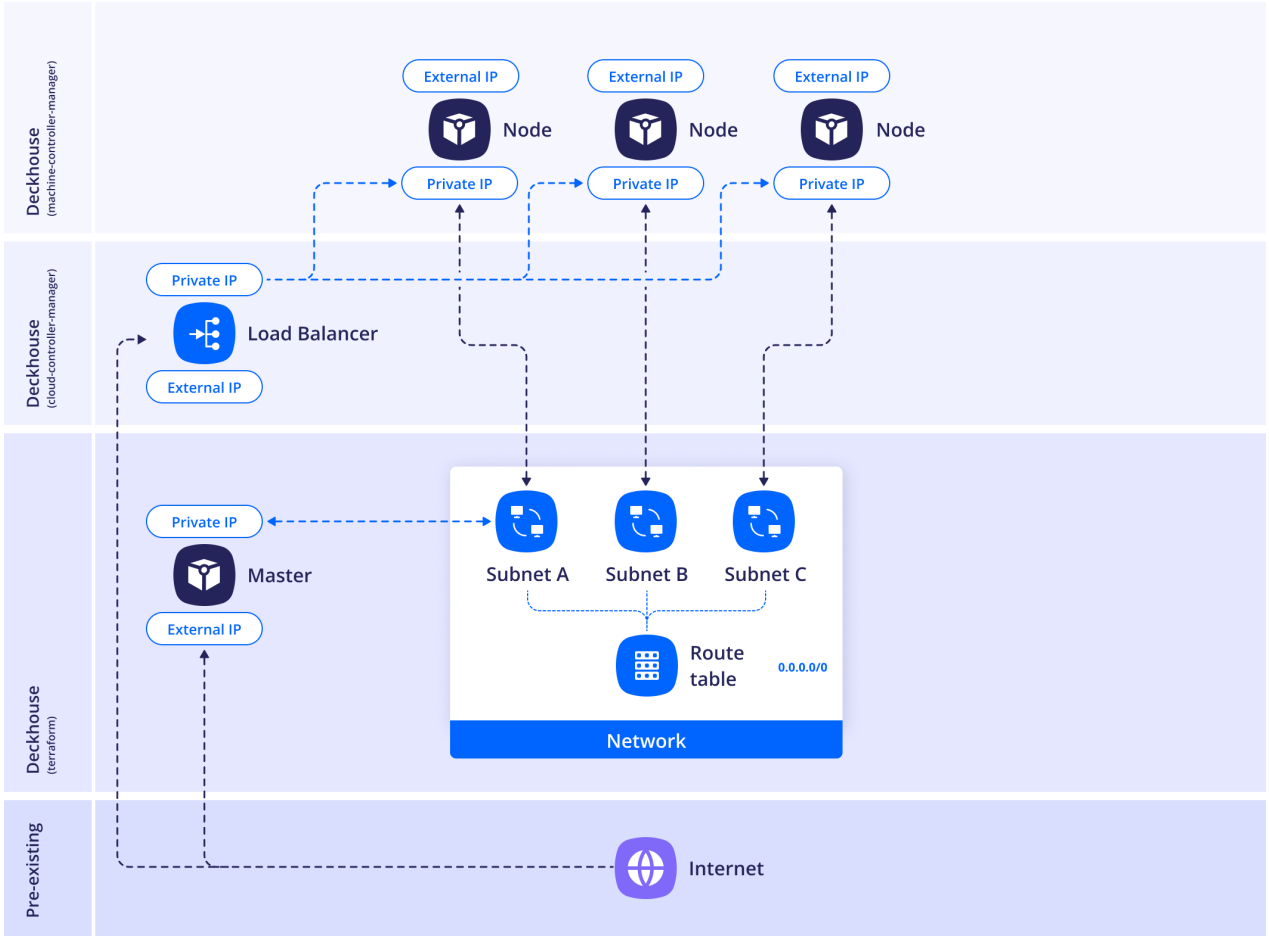
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 192.168.12.13/24
existingNetworkID: <EXISTING_NETWORK_ID>
provider:
  cloudID: <CLOUD_ID>
  folderID: <FOLDER_ID>
  serviceAccountJSON: |
    {
    "id": "id",
    "service_account_id": "service_account_id",
    "key_algorithm": "RSA_2048",
    "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID....AQAB\n-----END PUBLIC KEY-----\n",
    "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE....1ZPJeBLt+\n-----END PRIVATE KEY-----\n"
    }
masterNodeGroup:
  replicas: 3
  zones:
  - ru-central1-a
  - ru-central1-b
  - ru-central1-d
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    # Optional: uncomment to assign public IP addresses to master nodes.
    # externalIPAddresses:
    # - "<ZONE_A_EXTERNAL_IP_MASTER_1>"
    # - "Auto"
    # - "Auto"
    # externalSubnetIDs:
    # - <ZONE_A_SUBNET_ID>
    # - <ZONE_B_SUBNET_ID>
    # - <ZONE_D_SUBNET_ID>
    additionalLabels:
      takes: priority
nodeGroups:
- name: worker
  replicas: 2
  zones:
  - ru-central1-a
  - ru-central1-b
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    coreFraction: 50
    # Optional: uncomment to assign public IP addresses to worker nodes.
    # externalIPAddresses:
    # - "Auto"
    # - "Auto"
    # externalSubnetIDs:
    # - <ZONE_A_SUBNET_ID>
    # - <ZONE_B_SUBNET_ID>
    additionalLabels:
      role: example
labels:
  billing: prod
dhcpOptions:
  domainName: test.local
  domainNameServers:
  - <DNS_SERVER_1>
  - <DNS_SERVER_2>
```

WithoutNAT

In this layout, no form of NAT is used, and each node is assigned a public IP address.

⚠ DKP does not support security groups, so all cluster nodes will be accessible without connection restrictions.



Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: WithoutNAT
provider:
  cloudID: <CLOUD_ID>
  folderID: <FOLDER_ID>
  serviceAccountJSON: |
    {
    "id": "id",
    "service_account_id": "service_account_id",
    "key_algorithm": "RSA_2048",
    "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID....AQAB\n-----END PUBLIC KEY-----\n",
    "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE....1ZPJeBLt+\n-----END PRIVATE KEY-----\n"
    }
masterNodeGroup:
  replicas: 3
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    externalIPAddresses:
    - "Auto"
    - "Auto"
    - "Auto"
    externalSubnetIDs:
    - <ZONE_A_SUBNET_ID>
    - <ZONE_B_SUBNET_ID>
    - <ZONE_D_SUBNET_ID>
    zones:
    - ru-central1-a
    - ru-central1-b
    - ru-central1-d
nodeGroups:
- name: worker
  replicas: 2
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    coreFraction: 50
    externalIPAddresses:
    - "<ZONE_A_EXTERNAL_IP_WORKER_1>"
    - "Auto"
    externalSubnetIDs:
    - <ZONE_A_SUBNET_ID>
    - <ZONE_B_SUBNET_ID>
    zones:
    - ru-central1-a
    - ru-central1-b
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 192.168.12.13/24
existingNetworkID: <EXISTING_NETWORK_ID>
dhcpOptions:
  domainName: test.local
  domainNameServers:
  - <DNS_SERVER_1>
  - <DNS_SERVER_2>
```

WithNATInstance

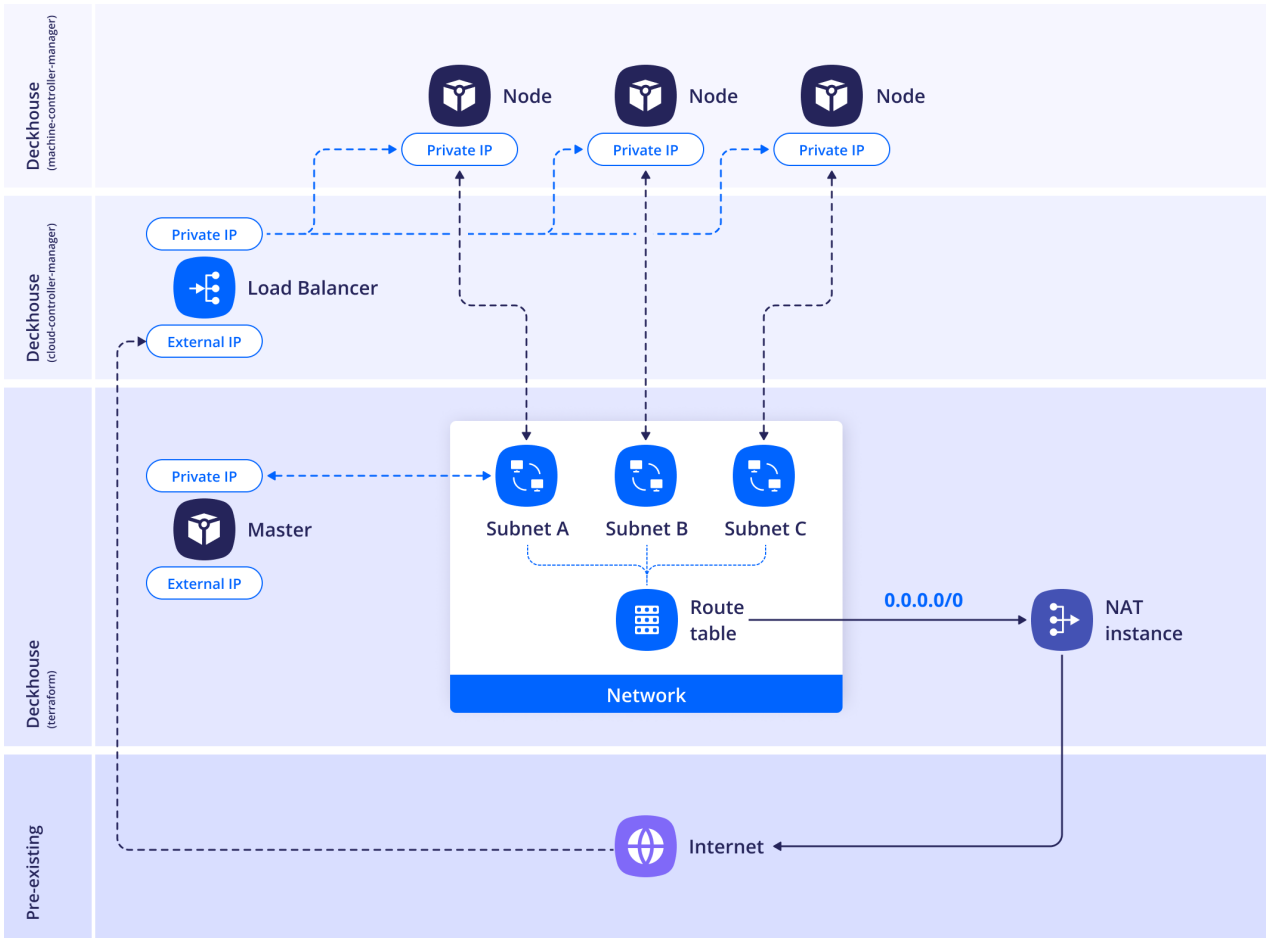
In this layout, a NAT instance is created in a separate subnet, and the route tables of the zone subnets are updated with a `0.0.0.0/0` route using the NAT instance as the next hop. The subnet is isolated to prevent routing loops and must not overlap with any other networks used in the cluster.

To place the NAT instance in an existing subnet, use the `withNATInstance.internalSubnetID` parameter. The instance will be created in the zone corresponding to this subnet.

If you need to create a new subnet, specify the `withNATInstance.internalSubnetCIDR` parameter. The NAT instance will be placed in this subnet.

> **i** One of the following parameters is required: `withNATInstance.internalSubnetID` or `withNATInstance.internalSubnetCIDR`.

If `withNATInstance.externalSubnetID` is also specified, the NAT instance will be connected to that subnet via a secondary network interface.



Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: WithNATInstance
withNATInstance:
  natInstanceExternalAddress: <NAT_INSTANCE_EXTERNAL_ADDRESS>
  internalSubnetID: <INTERNAL_SUBNET_ID>
  externalSubnetID: <EXTERNAL_SUBNET_ID>
provider:
  cloudID: <CLOUD_ID>
  folderID: <FOLDER_ID>
  serviceAccountJSON: |
    {
    "id": "id",
    "service_account_id": "service_account_id",
    "key_algorithm": "RSA_2048",
    "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID....AQAB\n-----END PUBLIC KEY-----\n",
    "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE....1ZPJeBLt+\n-----END PRIVATE KEY-----\n"
    }
masterNodeGroup:
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
nodeGroups:
- name: worker
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    coreFraction: 50
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 192.168.12.13/24
existingNetworkID: <EXISTING_NETWORK_ID>
dhcpOptions:
  domainName: test.local
  domainNameServers:
  - <DNS_SERVER_1>
  - <DNS_SERVER_2>
```

i

In this example, master and worker nodes are created without public IP addresses and without additional external network interfaces. Outbound traffic from the cluster subnets goes through the NAT instance.

The NAT instance is not automatically used by the `dhctl` installer as a bastion host or jump host. To install the cluster, the master node must be accessible over SSH from the machine where `dhctl` is running: either directly or through a bastion host.

If the master node does not have a public IP address or is not directly accessible, run the installation with the `--ssh-bastion-host` and `--ssh-bastion-user` parameters.

If you need to connect to the master node directly using a public IP address, specify the `externalIPAddresses` and `externalSubnetIDs` parameters for it. Keep in mind that the interface from the network specified in `externalSubnetIDs` will be used as the node's default gateway.

Defining the YandexClusterConfiguration

To integrate DKP with Yandex Cloud, you need to describe the cluster infrastructure using the YandexClusterConfiguration resource.

YandexClusterConfiguration is a custom resource (CR) that defines the parameters for integrating with Yandex Cloud. DKP uses this resource to:

- Deploy master and worker nodes in the cloud.
- Define the network layout.
- Connect to the Yandex Cloud API using an authorization JSON key.
- Configure subnets, public IP addresses, VM resources, and more.

Required fields:

- `apiVersion`: Must be `deckhouse.io/v1`.
- `kind`: Must be `YandexClusterConfiguration`.

Example resource header:

```
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
```

To edit this resource in a running cluster, execute the following command:

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

After making changes, apply them using the following command:

```
dhctl converge
```

Configuration example

Below is a minimal example of a YandexClusterConfiguration resource that defines a cluster with one master group and one worker node group. It uses the Standard layout and includes basic compute resource parameters, an SSH public key, and cloud and folder identifiers:

```
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: Standard
nodeNetworkCIDR: 127.0.0.1/8
labels:
  label-2: b
sshPublicKey: "<SSH_PUBLIC_KEY>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    cores: 4
    memory: 8192
    imageID: fd8nb7ecsbvj76dfaa8b
nodeGroups:
- name: worker
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    cores: 4
    memory: 8192
    imageID: fd8nb7ecsbvj76dfaa8b
    coreFraction: 50
    externalIPAddresses:
    - 198.51.100.5
    - Auto
provider:
  cloudID: "<CLOUD_ID>"
  folderID: "<FOLDER_ID>"
  serviceAccountJSON: |
    {
    "id": "id",
    "service_account_id": "service_account_id",
    "key_algorithm": "RSA_2048",
    "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID....AQAB\n-----END PUBLIC KEY-----\n",
    "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE....1ZPJeBLt+\n-----END PRIVATE KEY-----\n"
    }
```

- The master group consists of a single node, with no explicit zone specified.
- The worker group includes one node located in the `ru-central1-a` zone, with two external IP addresses: one manually specified (`198.51.100.5`), the other automatically allocated (`Auto`).
- The `serviceAccountJSON` field is specified for accessing the Yandex Cloud API.
- The configuration uses the `127.0.0.1/8` CIDR subnet and includes a cluster-level label `label-2: b`.

Network parameters and security

This section describes the settings related to addressing, routing, external traffic, and network security in a DKP cluster deployed in Yandex Cloud.

Internal node addressing

The `nodeNetworkCIDR` parameter defines the IP address range for splitting between availability zones and applying to the internal node interfaces.

```
nodeNetworkCIDR: 192.168.12.13/24
```

Depending on the selected layout (Standard or WithNATInstance), this subnet will be automatically divided into three equal parts for each availability zone:

- ru-central1-a
- ru-central1-b
- ru-central1-d

Each part will be used as a separate internal subnet to which the nodes created in the corresponding zone will be connected.

> **i** If you plan to use the same subnet across multiple clusters (for example, with `cni-simple-bridge`), keep in mind the limitation: one cluster = one routing table = one subnet. It impossible to deploy two clusters using `cni-simple-bridge` in the same subnet. If you need to reuse subnets across clusters, use `cni-cilium`.

Assigning external IP addresses and outgoing traffic

The `externalSubnetIDs` parameter is specified in the `masterNodeGroup.instanceClass` and `nodeGroups.instanceClass` sections. It is an array of Yandex Cloud subnet IDs used to attach external network interfaces to nodes. This parameter is required when:

- Assigning a public IP address.
- Defining a default route for outgoing traffic from nodes.
- Using `Auto` values in the `externalIPAddresses` field.

Example:

```
externalSubnetIDs:
  - <RU-CENTRAL1-A-SUBNET-ID>
  - <RU-CENTRAL1-B-SUBNET-ID>
  - <RU-CENTRAL1-D-SUBNET-ID>
```

> **i** The `externalSubnetIDs` parameter is required for the automatic assignment of public IPs using `externalIPAddresses: ["Auto", ...]`.

DNS and DHCP settings for internal networks

The `dhcpOptions` parameter lets you set the DHCP server configuration applied to all subnets created within the DKP cluster in Yandex Cloud.

Available fields:

- `domainName`: The search domain to be set in the network configuration.
- `domainNameServers`: An array of IP addresses of DNS servers to be used as recursive resolvers.

Example:

```
dhcpOptions:
  domainName: test.local
  domainNameServers:
  - 192.168.0.2
  - 192.168.0.3
```

> **i** The DNS servers listed in `domainNameServers` must resolve both external and internal domain zones used by the cluster. Otherwise, service disruptions may occur.

After updating `dhcpOptions`:

- Force renew the `DHCP lease` on all VMs.
- Restart all pods with `hostNetwork: true`, such as `kube-dns`, to refresh the `resolv.conf` contents.

You can apply changes using the following command:

```
netplan apply
```

or another appropriate method depending on your system (such as `systemd-networkd`, `dhclient`, etc.).

Using pre-existing subnets

The `existingZoneToSubnetIDMap` parameter lets you specify mappings between availability zones and pre-created subnets in Yandex Cloud. This is important if you don’t want DKP to automatically create subnets and prefer to use existing ones.

Example:

```
existingZoneToSubnetIDMap:
  ru-central1-a: e2lu8r1tbbtryhdpa9ro
  ru-central1-b: e2lu8r1tbbtryhdpa9ro
  ru-central1-d: e2lu8r1tbbtryhdpa9ro
```

i DKP creates a routing table automatically but does not associate it with subnets. You need to do that manually via the Yandex Cloud interface.

Additional external networks

DKP allows you to explicitly specify a list of additional external networks whose IP addresses will be treated as public (external IPs). This is configured via the `settings.additionalExternalNetworkIDs` parameter in the ModuleConfig resource.

This is useful when:

- There are external subnets not listed in `externalSubnetIDs` but that should be treated as external.
- You need precise control over how IP addresses are interpreted for load balancing, routing, and status export.
- You use custom network configurations (for example, external NAT gateways or manually reserved IPs).

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 1
  enabled: true
  settings:
    additionalExternalNetworkIDs:
      - enp6t4sno
```

If this parameter is not set, DKP will only use the subnets explicitly specified in the YandexClusterConfiguration (for example, via `externalSubnetIDs`) to determine whether an IP is public.

Configuring security groups in Yandex Cloud

When a cloud network is created, Yandex Cloud automatically adds a default security group for all attached networks, including the one used by the DKP cluster. This default group includes rules that allow all incoming and outgoing traffic. It applies to all subnets in the cloud network unless a different security group is explicitly assigned to a VM interface.

✖ Do not remove the default rules allowing all traffic until your security group configuration is fully completed. Doing so may disrupt cluster functionality.

Below are general guidelines for configuring security groups. Incorrectly configured security group rules may negatively impact on cluster operation. Read through the security group specifics in Yandex Cloud before using them in production environments.

1. Identify the cloud network used by the DKP cluster.

   The network name matches the `prefix` field in the ClusterConfiguration resource. You can retrieve it using the following command:

   ```
   kubectl get secrets -n kube-system d8-cluster-configuration -ojson | \
     jq -r '.data."cluster-configuration.yaml"' | base64 -d | grep prefix | cut -d: -f2
   ```

2. In the Yandex Cloud Console, go to the Virtual Private Cloud service and open the Security Groups section. You should see a security group marked `Default`.
3. Create rules according to the Yandex Cloud instructions.
4. Delete the rule that allows all **incoming** traffic and save your changes.

Configuring access via bastion host

To connect to nodes located in private subnets (for example, when using the Standard or WithNATInstance layout), you can use a bastion host — an intermediary VM with a public IP address that provides SSH access to the nodes.

To configure access, follow these steps:

1. Bootstrap the base infrastructure. Before creating a bastion host, perform the initial installation phase of DKP, which sets up the network infrastructure:

   ```
   dhctl bootstrap-phase base-infra --config config.yml
   ```

2. Create a bastion host in Yandex Cloud:

```
yc compute instance create \
  --name bastion \
  --hostname bastion \
  --create-boot-disk image-family=ubuntu-2204-lts,image-folder-id=standard-images,size=20,type=network-hdd \
  --memory 2 \
  --cores 2 \
  --core-fraction 100 \
  --ssh-key ~/.ssh/<SSH_PUBLIC_KEY_FILE> \
  --zone ru-central1-a \
  --public-address 178.154.226.159
```

> **i** Replace `<SSH_PUBLIC KEY FILE>` here with the name of your public key. For example, for a key with RSA encryption, it will be `id_rsa.pub`, and for a key with ED25519 encryption, it will be with `id_ed25519.pub`.

Ensure the IP address specified in `--public-address` is accessible from your network and is correct.

3. Start the main DKP bootstrap through the bastion host:

```
dhctl bootstrap --ssh-bastion-host=178.154.226.159 --ssh-bastion-user=yc-user \
  --ssh-user=ubuntu --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> --config=/config.yml
```

Where:

- `--ssh-bastion-user`: User for connecting to the bastion host.
- `--ssh-user`: User on the target cluster nodes.
- `--ssh-agent-private-keys`: Path to the private SSH key.
- `--config`: Path to the DKP configuration file.

Additional resources

- Global custom resources
- Module `cloud-provider-yandex` parameters
- Module `cloud-provider-yandex` provider configuration
- Module `cni-cilium` documentation
- Module `cni-simple-bridge` documentation

Storage and load balancing in Yandex Cloud

This section covers additional aspects of Deckhouse Kubernetes Platform (DKP) integration with Yandex Cloud:

- Attaching cloud disks via CSI.
- Automatic StorageClass creation.
- Use of load balancers.
- Specifics of applying changes.
- Working with CloudStatic nodes and bastion hosts.

Storage (CSI and StorageClass)

DKP integrates with Yandex Cloud block storage via the Container Storage Interface (CSI) component. This allows DKP clusters to automatically provision and attach disks and use standard Kubernetes PersistentVolumeClaim resources for working with the storage.

DKP automatically creates StorageClass resources for all supported Yandex Cloud disk types, enabling immediate storage usage without the need to define classes manually.

The following disk types are supported:

Disk type	StorageClass name	Notes
`network-hdd`	`network-hdd`	—
`network-ssd`	`network-ssd`	—
`network-ssd-nonreplicated`	`network-ssd-nonreplicated`	Size must be a multiple of 93 GB
`network-ssd-io-m3`	`network-ssd-io-m3`	Size must be a multiple of 93 GB

> **i** The sizes of `network-ssd-nonreplicated` and `network-ssd-io-m3` disks must be multiples of 93 GB. Otherwise, volume provisioning will fail.

Excluding unnecessary StorageClasses

If certain disk types won't be used in the cluster, you can disable automatic creation of the corresponding StorageClass objects using the `settings.storageClass.exclude` parameter in the ModuleConfig resource:

```
settings:
  storageClass:
    exclude:
    - network-ssd-.*
    - network-hdd
```

In this example, DKP will not create StorageClass resources for any `network-ssd` or `network-hdd` disks.

Setting the default StorageClass

By default, DKP determines the StorageClass using the `storageclass.kubernetes.io/is-default-class=true` annotation.

To explicitly set a different default StorageClass, use the global DKP parameter `global.defaultClusterStorageClass`. You can modify it with the following command:

```
kubectl edit mc global
```

If `defaultClusterStorageClass` is not specified, DKP determines the default StorageClass in the following order:

- A StorageClass with the annotation `storageclass.kubernetes.io/is-default-class='true'` (if it exists in the cluster).
- The first StorageClass in alphabetical order auto-created by the cloud provider.
- The default value of `defaultClusterStorageClass` is an empty string (`""`).

Resizing a PVC

You can increase the size of an existing PVC by changing the value of the `spec.resources.requests.storage` parameter, without stopping or recreating the pod that uses it.

After the value of `spec.resources.requests.storage` is changed, the CSI driver performs the following operations:

- increases the size of the disk in Yandex Cloud;
- updates the size of the associated PersistentVolume;
- expands the file system on the node to which the volume is attached.

During the operation, the pod continues running, and the mounted volume remains available to the application. After the resize operation is complete, the new file system size becomes available inside the container without restarting the pod.

> **i** Decreasing the size of a PVC is not supported.

To increase the size of a PVC, follow these steps:

1. Get the name of the StorageClass used by the PVC:

   ```
   d8 k -n <NAMESPACE> get pvc <PVC_NAME> \
     -o jsonpath='{.spec.storageClassName}{"\n"}'
   ```

   Where:

   - `<NAMESPACE>` — the namespace containing the PVC;
   - `<PVC_NAME>` — the name of the PVC to resize.

   For example:

   ```
   d8 k -n production get pvc application-data \
     -o jsonpath='{.spec.storageClassName}{"\n"}'
   ```

   Example output:

   ```
   network-ssd
   ```

   In this example, the `application-data` PVC uses the `network-ssd` StorageClass.

2. Make sure that the StorageClass allows volume expansion:

   ```
   d8 k get storageclass <STORAGECLASS_NAME> \
     -o jsonpath='{.allowVolumeExpansion}{"\n"}'
   ```

   Where `<STORAGECLASS_NAME>` is the name of the StorageClass obtained in the previous step.

   For example:

   ```
   d8 k get storageclass network-ssd \
     -o jsonpath='{.allowVolumeExpansion}{"\n"}'
   ```

   Example output:

```
true
```

3. Check the current status and size of the PVC:

   ```
   d8 k -n <NAMESPACE> get pvc <PVC_NAME>
   ```

   For example:

   ```
   d8 k -n production get pvc application-data
   ```

   Example output:

   ```
   NAME               STATUS   VOLUME                                     CAPACITY   ACCESS MODES
   STORAGECLASS
   application-data   Bound    pvc-65e92674-077c-4b4f-b65d-19e92f04e103   20Gi       RWO
   network-ssd
   ```

   Make sure that:

   - the PVC is in the `Bound` state;
   - the `CAPACITY` field shows the current PVC size;
   - the `STORAGECLASS` field shows the StorageClass checked in the previous step.

4. Increase the PVC size:

   ```
   d8 k -n <NAMESPACE> edit pvc <PVC_NAME>
   ```

   For example:

   ```
   d8 k -n production edit pvc application-data
   ```

   Specify the new PVC size in the `spec.resources.requests.storage` field:

   ```
   spec:
     resources:
       requests:
         storage: 30Gi
   ```

   In this example, the PVC size is increased to `30Gi`.

   Save the changes and close the editor.

   > ⚠ For the `network-ssd-nonreplicated` and `network-ssd-io-m3` StorageClasses, the size must be a multiple of 93Gi.

5. Wait for the PVC resize operation to complete:

   ```
   d8 k -n <NAMESPACE> get pvc <PVC_NAME> --watch
   ```

   Where:

   - `<NAMESPACE>` — the namespace containing the PVC;
   - `<PVC_NAME>` — the name of the PVC being resized.

   For example:

   ```
   d8 k -n production get pvc application-data --watch
   ```

   During the resize operation, the `CAPACITY` field may continue to show the previous size:

   ```
   NAME               STATUS   VOLUME                                     CAPACITY   ACCESS MODES
   STORAGECLASS
   application-data   Bound    pvc-65e92674-077c-4b4f-b65d-19e92f04e103   20Gi       RWO
   network-ssd
   ```

   The operation is complete when the `CAPACITY` field shows the new PVC size:

   ```
   NAME               STATUS   VOLUME                                     CAPACITY   ACCESS MODES
   STORAGECLASS
   application-data   Bound    pvc-65e92674-077c-4b4f-b65d-19e92f04e103   30Gi       RWO
   network-ssd
   ```

6. Check the PVC events:

   ```
   d8 k -n <NAMESPACE> describe pvc <PVC_NAME>
   ```

   For example:

   ```
   d8 k -n production describe pvc application-data
   ```

   The following events may appear during the resize operation:

   ```
   ExternalExpanding
   Resizing
   FileSystemResizeRequired
   ```

   The following event indicates that the file system was successfully expanded:

   ```
   FileSystemResizeSuccessful
   ```

   For example:

   ```
   Normal  FileSystemResizeSuccessful  kubelet  MountVolume.NodeExpandVolume succeeded for volume
   "pvc-65e92674-077c-4b4f-b65d-19e92f04e103"
   ```

7. Check the file system size inside the pod:

   ```
   d8 k -n <NAMESPACE> exec <POD_NAME> -- \
     df -hT <MOUNT_PATH>
   ```

   Where:

   - `<NAMESPACE>` — the namespace containing the pod;
   - `<POD_NAME>` — the name of the pod using the PVC;
   - `<MOUNT_PATH>` — the path inside the container where the PVC is mounted.

   For example:

   ```
   d8 k -n production exec application-0 -- \
     df -hT /data
   ```

   Example output:

   ```
   Filesystem   Type   Size    Used   Avail   Use%   Mounted on
   /dev/vde     ext4   29.4G   22M    29.4G    1%    /data
   ```

   > **i** The actual file system size may be slightly smaller than the PVC size due to file system metadata.

Load balancing

External LoadBalancer

DKP automatically watches for Kubernetes Service objects of type LoadBalancer. When such a service is created, DKP creates the following resources:

- **NetworkLoadBalancer**: A Yandex Cloud network load balancer.
- **TargetGroup**: A group of endpoints for traffic distribution.

These resources allow Kubernetes LoadBalancer services to receive traffic from the internet or internal networks, depending on the configuration.

For more details on the architecture, refer to the Kubernetes Cloud Controller Manager for Yandex Cloud documentation .

Internal LoadBalancer

To create an internal load balancer, you must explicitly specify the subnet where the load balancer listener should be created.

To do this, add the following annotation to the Service object:

```
metadata:
  annotations:
    yandex.cpi.flant.com/listener-subnet-id: <SubnetID>
```

`SubnetID` refers to the ID of the subnet where the internal listener for the Yandex LoadBalancer will be created. This allows you to control the load balancer's network exposure and limit it to internal addresses only.

> **i** The default behavior (external or internal LB) depends on the cluster configuration. To explicitly choose the type, use the `yandex.cpi.flant.com/loadbalancer-external` annotation.

Service annotations

Default values are configured in the cluster for placing load balancer resources (the network for the target group and the subnet for the Listener). These values are set automatically during cluster setup and can be overridden with annotations at the individual Service level.

The following annotations are supported by Yandex Cloud Controller Manager:

1. `yandex.cpi.flant.com/target-group-network-id` — specifies the NetworkID in which the target group for this Service will be created. Overrides the corresponding default value.
2. `yandex.cpi.flant.com/listener-subnet-id` — sets the SubnetID for the Listeners of the LB created for this Service. Overrides the corresponding default value.
3. `yandex.cpi.flant.com/listener-address-ipv4` — sets a predefined IPv4 address for the Listeners (supported for both internal and external LBs).
4. `yandex.cpi.flant.com/loadbalancer-external` — enables creation of an external LB for this Service (use it when you need to explicitly create an external load balancer). Overrides the default behavior.
5. `yandex.cpi.flant.com/target-group-name-prefix` — specifies the name prefix of the target group used by the LoadBalancer. The annotation on the Service identifies the target group but does not select nodes. To include nodes in the target group, set the same annotation value in `spec.nodeTemplate.annotations` in the NodeGroup. Yandex Cloud Controller Manager includes all suitable nodes carrying this annotation value, regardless of their NodeGroup. The target group name is formed as `<ANNOTATION_VALUE><YANDEX_CLOUD_CLUSTER_NAME><NETWORK_ID>`.

If separate target groups are created for the control plane or master nodes, add the label `node.kubernetes.io/exclude-from-external-load-balancers: ""` to the master nodes. This prevents the controller from automatically adding master nodes to new target groups for load balancers. If you create your own load balancer for master nodes and want YCC to also be able to place its load balancers on master nodes, pre-create a target group with a name matching the pattern `${CLUSTER-NAME}${VPC.ID}`.

Using a separate target group for a NodeGroup

By default, Yandex Cloud Controller Manager adds all suitable cluster nodes to the default target group. To assign nodes to a separate target group, use the `yandex.cpi.flant.com/target-group-name-prefix` annotation.

1. In the NodeGroup whose nodes should be included in a separate target group, specify the `yandex.cpi.flant.com/target-group-name-prefix` annotation in the `spec.nodeTemplate.annotations` parameter. For example:

   ```
   spec:
     nodeTemplate:
       annotations:
         yandex.cpi.flant.com/target-group-name-prefix: frontend-
   ```

   The annotation is propagated to the Node objects. Yandex Cloud Controller Manager adds all suitable nodes carrying the same annotation value to the target group, regardless of their NodeGroup.

2. In the Service object of the LoadBalancer type, specify the same annotation with the same value. For example:

   ```
   metadata:
     annotations:
       yandex.cpi.flant.com/target-group-name-prefix: frontend-
   ```

   The annotation on the Service identifies the target group used by the LoadBalancer but does not select nodes.

3. Make sure that the values of `yandex.cpi.flant.com/target-group-name-prefix` in the NodeGroup and Service match. In the example above, both resources use the `frontend-` value.

4. The target group will be created with a name generated according to the following pattern:

   ```
   <ANNOTATION_VALUE><YANDEX_CLOUD_CLUSTER_NAME><NETWORK_ID>
   ```

   For example, if the annotation value is `frontend-`, the cluster name is `my-cluster-`, and the network ID is `enp123456789`, the target group name will be:

   ```
   frontend-my-cluster-enp123456789
   ```

Configuration example:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: frontend
spec:
  nodeType: CloudEphemeral
  nodeTemplate:
    annotations:
      yandex.cpi.flant.com/target-group-name-prefix: frontend-
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-frontend
  annotations:
    yandex.cpi.flant.com/target-group-name-prefix: frontend-
spec:
  type: LoadBalancer
```

⚠ Yandex Cloud does not allow a single target, identified by the (`SubnetID, IP`) pair, to belong to multiple target groups at the same time.

When the target group is changed, Yandex Cloud Controller Manager first removes the target from the current group and then adds it to the new one. This transition may cause a brief interruption in traffic handling.

Nodes with a dedicated target group prefix are excluded from the default target group. As a result, load balancers that use the default target group stop routing traffic to those nodes.

This automatic transition is performed only between target groups managed by Yandex Cloud Controller Manager. If a target already belongs to another target group that is not managed by Yandex Cloud Controller Manager, remove it from that group before using the annotation.

After the prefix is changed or removed, the previous target group may remain empty. If it is no longer used by any load balancers, delete it manually.

Target group health checks

Health check parameters (for LB target groups created by the controller):

1. `yandex.cpi.flant.com/healthcheck-interval-seconds` — how often to run the check, in seconds (default: 2).
2. `yandex.cpi.flant.com/healthcheck-timeout-seconds` — how long to wait for an endpoint response, in seconds. If no response is received within this time, the check is considered failed (default: 1).
3. `yandex.cpi.flant.com/healthcheck-unhealthy-threshold` — how many consecutive failed checks are required to mark an endpoint as unhealthy and exclude it from load balancing (default: 2).
4. `yandex.cpi.flant.com/healthcheck-healthy-threshold` — how many consecutive successful checks are required to return an endpoint to healthy status and include it back in load balancing (default: 2).

Applying changes

DKP does not recreate existing Machine objects when configuration parameters change. Node recreation occurs only when:

- NodeGroup section parameters change.
- YandexInstanceClass parameters change.

This behavior helps prevent unnecessary operations and node idling, but it means you must manually recreate VMs if needed.

If you change the YandexClusterConfiguration resource (for example, change provider settings, layout, subnets, etc.), run the following command to apply the changes:

```
dhctl converge
```

This command reconciles the cluster state with the configuration defined in the resources.

Changing `networkType` does not recreate CloudEphemeral nodes

In clusters that use Machine Controller Manager (MCM), changing the `networkType` parameter in a YandexInstanceClass resource does not trigger automatic recreation of existing CloudEphemeral nodes. Although the MachineClass is updated, the network acceleration type of existing virtual machines in Yandex Cloud remains unchanged.

DKP intentionally does not take the `networkType` parameter into account when determining whether CloudEphemeral nodes should be recreated in MCM. If it did, updating DKP would recreate CloudEphemeral nodes in all clusters where `networkType` is specified, even if its value had not changed.

In Cluster API, the `networkType` parameter is taken into account when determining whether nodes should be recreated, so changing it automatically triggers node recreation. New CloudEphemeral NodeGroups in Yandex Cloud use Cluster API by default.

If you changed `networkType` in an MCM-based cluster and want the new value to apply to existing CloudEphemeral

nodes, recreate them manually — instructions in the documentation.

Integrating manually created VMs

DKP allows you to connect existing VMs in Yandex Cloud to the Kubernetes cluster as nodes. These nodes are called CloudStatic and they are not directly managed by the `node-manager` module but can still be used in the cluster.

To manually connect a VM as a CloudStatic node, follow these steps:

1. Retrieve the current `nodeNetworkCIDR` value from the cluster:

   ```
   kubectl -n kube-system get secret d8-provider-cluster-configuration -o json | \
     jq --raw-output '.data."cloud-provider-cluster-configuration.yaml"' | base64 -d | grep
   '^nodeNetworkCIDR'
   ```

   Expected output:

   ```
   nodeNetworkCIDR: 192.168.12.13/24
   ```

   Copy this value and use it in the VM metadata as `value`.

2. Set the `node-network-cidr` parameter in the VM metadata:

   ```
   key: node-network-cidr
   value: <nodeNetworkCIDR from the cluster>
   ```

   The `node-network-cidr` parameter must match the value specified in the YandexClusterConfiguration resource under `nodeNetworkCIDR`.

Additional resources

- Global parameters
- Module `cloud-provider-yandex` documentation
- Module `node-manager` documentation

Integration with Yandex Cloud services

Deckhouse Kubernetes Platform (DKP) supports native integration with several Yandex Cloud services. This allows for external monitoring, secure secret management, and automated data synchronization between the infrastructure and the cluster.

This section describes steps to configure the following integrations:

- Yandex Lockbox via External Secrets Operator
- Yandex Managed Service for Prometheus

Integration with Yandex Lockbox

DKP supports integration with Yandex Lockbox using the External Secrets Operator (ESO). This allows Kubernetes secrets in the cluster to be automatically synchronized with Lockbox secrets.

To set up the integration:

1. Create a service account for External Secrets Operator:

   ```
   yc iam service-account create --name eso-service-account
   ```

2. Create an authorized key for the service account and save it to a file:

   ```
   yc iam key create --service-account-name eso-service-account --output authorized-key.json
   ```

3. Assign `lockbox.editor`, `lockbox.payloadViewer`, and `kms.keys.encrypterDecrypter` roles to the service account for accessing all secrets in the folder:

   ```
   folder_id=<folderID>
   yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --
   role lockbox.editor
   yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --
   role lockbox.payloadViewer
   yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --
   role kms.keys.encrypterDecrypter
   ```

   For fine-grained access control, refer to the Yandex Lockbox access documentation.

4. Install External Secrets Operator using the Helm chart as described in this guide.

   Note that you may need to specify additional parameters such as `nodeSelector`, `tolerations`, etc. To do this, use the `./external-secrets/values.yaml` file after extracting the Helm chart.

Download and extract the chart using the following command:

```
helm pull oci://cr.yandex/yc-marketplace/yandex-cloud/external-secrets/chart/external-secrets \
  --version 0.5.5 \
  --untar
```

Install the chart:

```
helm install -n external-secrets --create-namespace \
  --set-file auth.json=authorized-key.json \
  external-secrets ./external-secrets/
```

Where:

- `authorized-key.json`: The file created in step 2.

5. Create a SecretStore resource containing the `sa-creds` secret:

```
kubectl -n external-secrets apply -f - <<EOF

apiVersion: external-secrets.io/v1alpha1
kind: SecretStore
metadata:
  name: secret-store
spec:
  provider:
    yandexlockbox:
      auth:
        authorizedKeySecretRef:
          name: sa-creds
          key: key
EOF
```

Where:

- `sa-creds`: Name of the Secret containing the authorized key. The secret appears after the Helm chart is installed.
- `key`: Name of the key in the `.data` field of the secret.

6. Verify that External Secrets Operator is working:

```
kubectl -n external-secrets get po
```

Example output:

```
NAME                                                READY   STATUS    RESTARTS   AGE
external-secrets-55f78c44cf-dbf6q                   1/1     Running   0          77m
external-secrets-cert-controller-78cbc7d9c8-rszhx   1/1     Running   0          77m
external-secrets-webhook-6d7b66758-s7v9c            1/1     Running   0          77m
```

Check the SecretStore status as well:

```
kubectl -n external-secrets get secretstores.external-secrets.io
```

Example output:

```
NAME           AGE   STATUS
secret-store   69m   Valid
```

7. Create a secret in Yandex Lockbox with the following parameters:
   - Name: `lockbox-secret`.
   - Key: Enter non-confidential ID `password`.
   - Value: Enter confidential data for storing `p@$$w0rd`.
8. Create an ExternalSecret pointing to the `lockbox-secret` in the `secret-store`:

```
kubectl -n external-secrets apply -f - <<EOF

apiVersion: external-secrets.io/v1alpha1
kind: ExternalSecret
metadata:
  name: external-secret
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: secret-store
    kind: SecretStore
  target:
    name: k8s-secret
  data:
  - secretKey: password
    remoteRef:
      key: <SECRET_ID>
      property: password
EOF
```

Where:

- `target.name`: Name of the new secret. External Secret Operator will create this secret in the DKP cluster and put the `lockbox-secret` Yandex Lockbox secret parameters into it.
- `data[].secretKey`: Name of the key in the `.data` field of the secret created by External Secret Operator.
- `data[].remoteRef.key`: ID of the `lockbox-secret` Yandex Lockbox secret created earlier (for example, `e6q28nvfmhu539******`).
- `data[].remoteRef.property`: Key for the `lockbox-secret` Yandex Lockbox secret.

9. Verify that the new `k8s-secret` key contains the `lockbox-secret` secret value:

```
kubectl -n external-secrets get secret k8s-secret -ojson | jq -r '.data.password' | base64 -d
```

The output will contain the `password` key value of the `lockbox-secret` secret created earlier:

```
p@$$w0rd
```

Integration with Yandex Managed Service for Prometheus

This integration lets you use Yandex Managed Service for Prometheus as an external storage for metrics (for example, for long-term retention).

To configure PrometheusRemoteWrite, follow these steps:

1. Create a service account with the `monitoring.editor` role.
2. Create an API key for the service account.
3. Create a PrometheusRemoteWrite resource:

```
kubectl apply -f - <<EOF

apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: yc-remote-write
spec:
  url: <REMOTE_WRITE_URL>
  bearerToken: <API_KEY>
EOF
```

Where:

- `<REMOTE_WRITE_URL>`: URL from the **Yandex Monitoring** -> **Prometheus** -> **Remote write** page.
- `<API_KEY>`: API key from the previous step (for example, `AQVN1HHJReSrfo9jU3aopsXrJyfq_UHs********`).

You can specify additional parameters as described in the `prometheus` module documentation.

For more details on writing metrics, refer to the Yandex Cloud documentation.

To read metrics in Grafana:

1. Create a service account with the `monitoring.viewer` role.
2. Create an API key for the service account.
3. Create a GrafanaAdditionalDatasource resource:

```
apiVersion: deckhouse.io/v1
kind: GrafanaAdditionalDatasource
metadata:
  name: managed-prometheus
spec:
  type: prometheus
  access: Proxy
  url: <GRAFANA_READ_URL>
  basicAuth: false
  jsonData:
    timeInterval: 30s
    httpMethod: POST
    httpHeaderName1: Authorization
  secureJsonData:
    httpHeaderValue1: Bearer <API_KEY>
```

Where:

- `<GRAFANA_READ_URL>`: URL from the **Yandex Monitoring** -> **Prometheus** -> **Reading Grafana metrics** page.
- `<API_KEY>`: API key from the previous step (for example, `AQVN1HHJReSrfo9jU3aopsXrJyfq_UHs********`).

You can specify additional parameters as described in the `prometheus` module documentation.

For more details on reading metrics with Grafana, refer to the Yandex Cloud documentation.

Additional resources

- Module `prometheus` custom resources

Connection and authorization in VK Cloud

To manage resources in VK Cloud using Deckhouse Kubernetes Platform, you must connect to the VK Cloud API.

The list of VK Cloud API services required for deployment is available in the Configuration section.

The user credentials required to access the VK Cloud API are provided in the OpenStack RC file (openrc file).

Instructions for downloading the openrc file via the standard web interface of VK Cloud and using it are available in the VK Cloud documentation.

If you're using a cloud provider's implementation of the VK Cloud API, the process for obtaining the openrc file may differ.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Layouts and configuration in VK Cloud

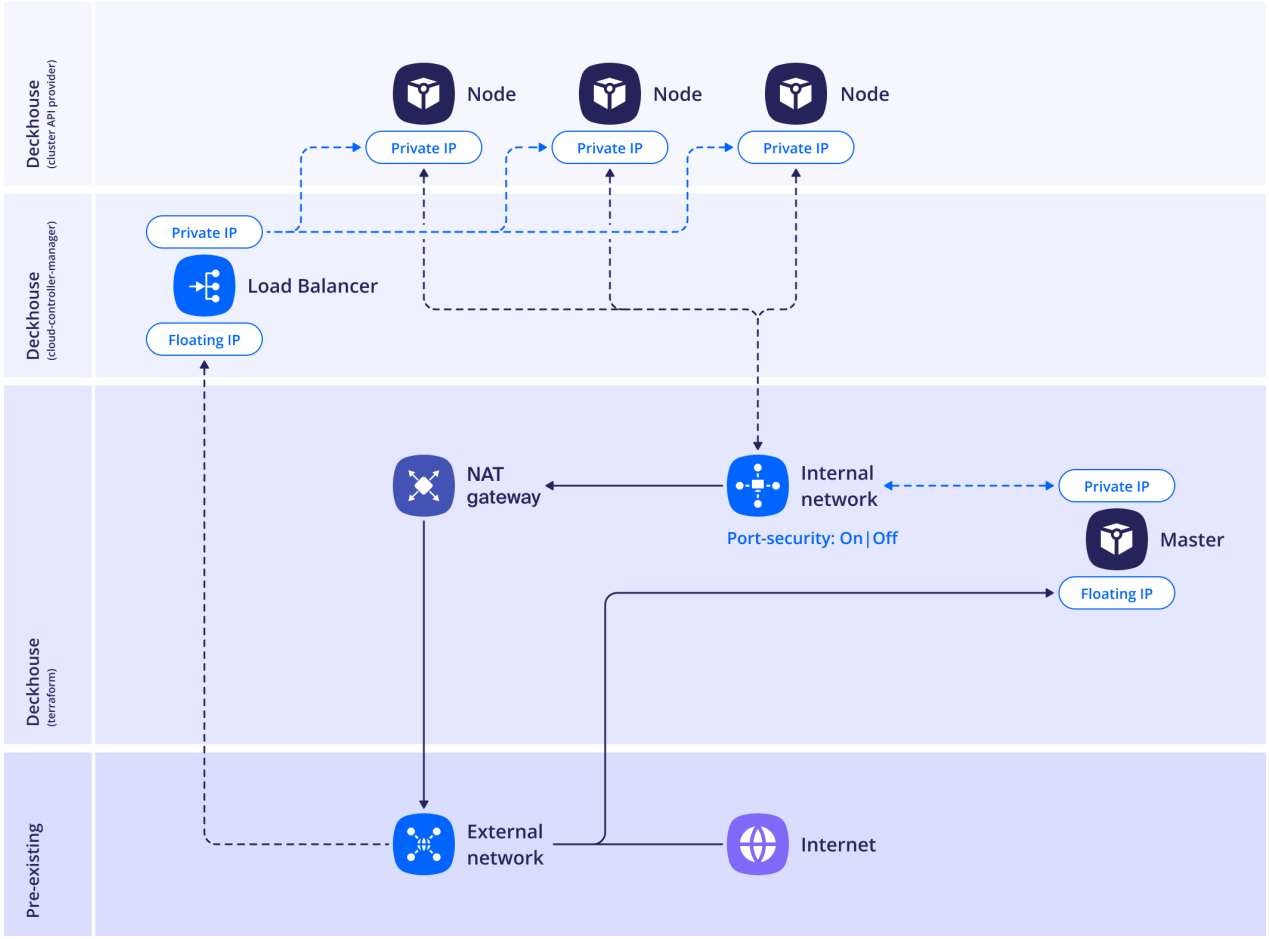
Layouts

This section describes the possible node placement layouts in VK Cloud infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. The floating IP is assigned to the master node.

⚠ If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port `6443`. To avoid this, we recommend using the SimpleWithInternalNetwork or Standard layout with a bastion host.



Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                 # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, true by default.
  externalNetworkName: shared                # Required.
  bastion:
    zone: ru2-b                             # Optional.
    volumeType: fast-ru-2b                   # Optional.
    instanceClass:
      flavorName: m1.large                   # Required.
      imageName: ubuntu-20-04-cloud-amd64      # Required.
      rootDiskSize: 50                      # Optional, 50 GB by default.
      additionalTags:
        severity: critical                   # Optional.
        environment: production              # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
```

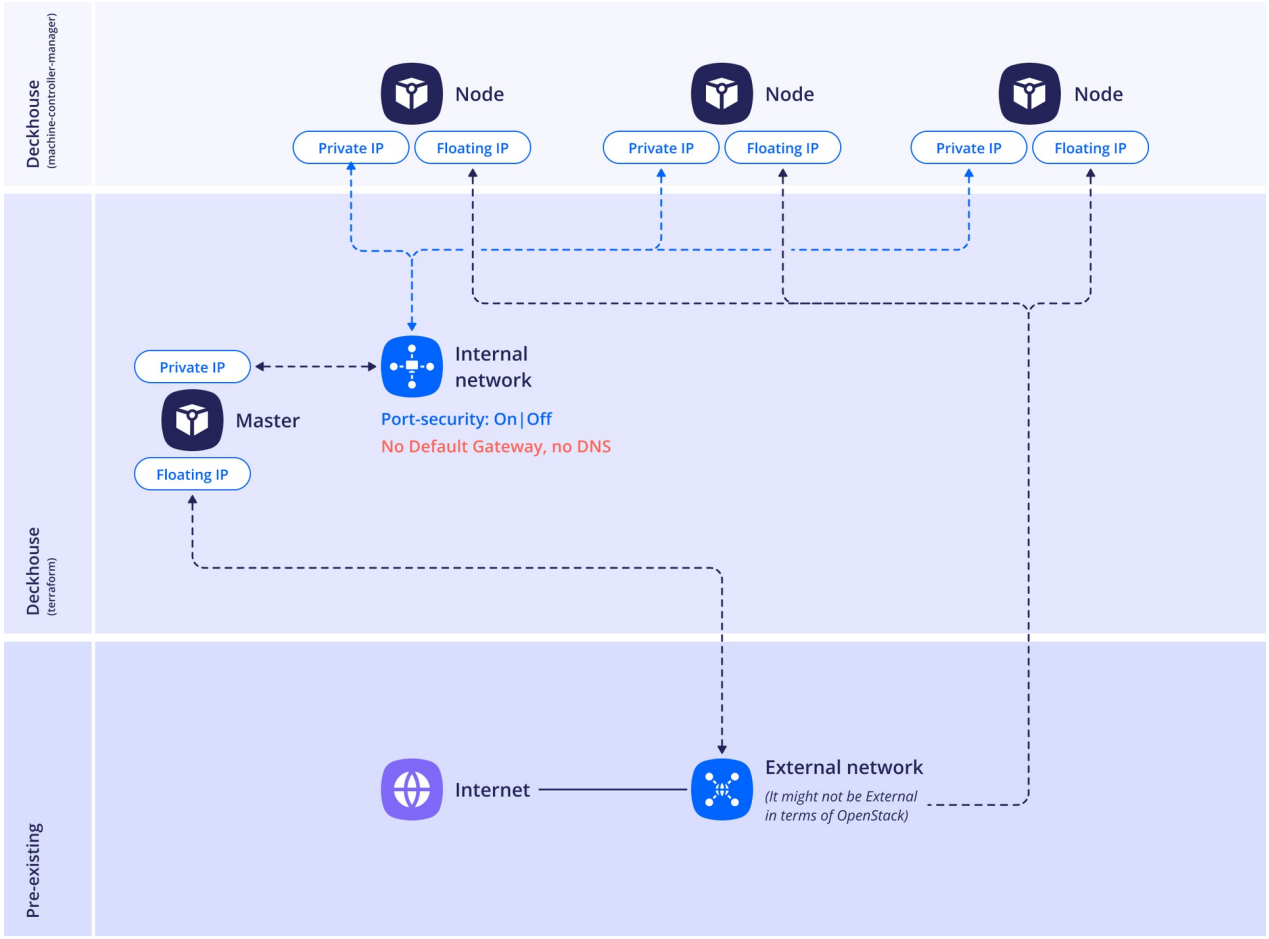
```
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                  # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default. Determines if configuration drive is
    # required during vm bootstrap process. It's needed if there
    # is no DHCP in the network that is used as a default gateway.
    configDrive: false
    # Required, the gateway of this network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                   # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

⚠

- This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.
- Using this strategy, make sure to explicitly specify the name of the internal network in `additionalNetworks` when creating an OpenStackInstanceClass in the cluster.



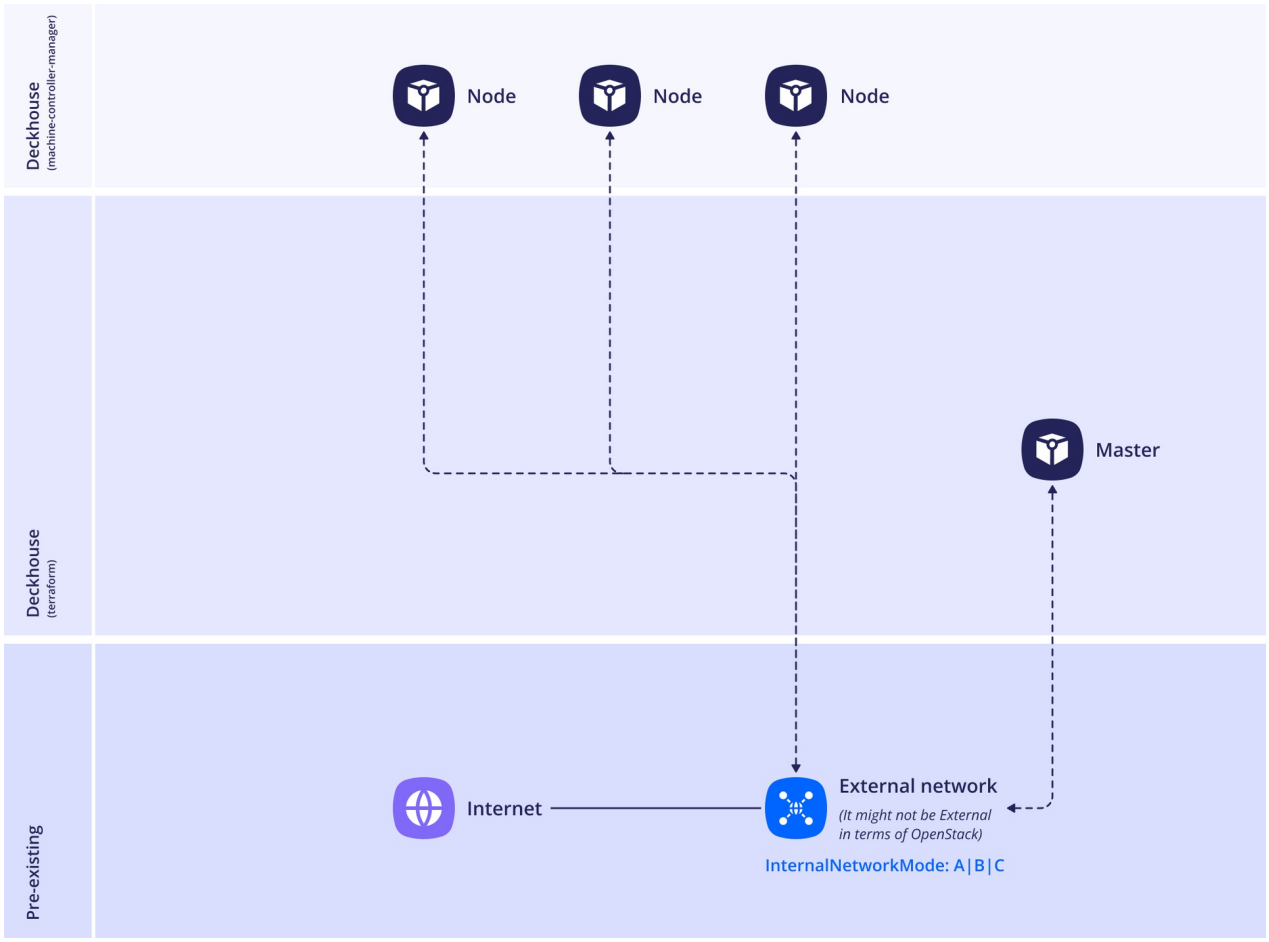
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, defines whether DHCP is enabled in specified external
  # network (true by default).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, true by default.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy can be useful if you need to merge a Kubernetes cluster with existing VMs.

⚠ This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.



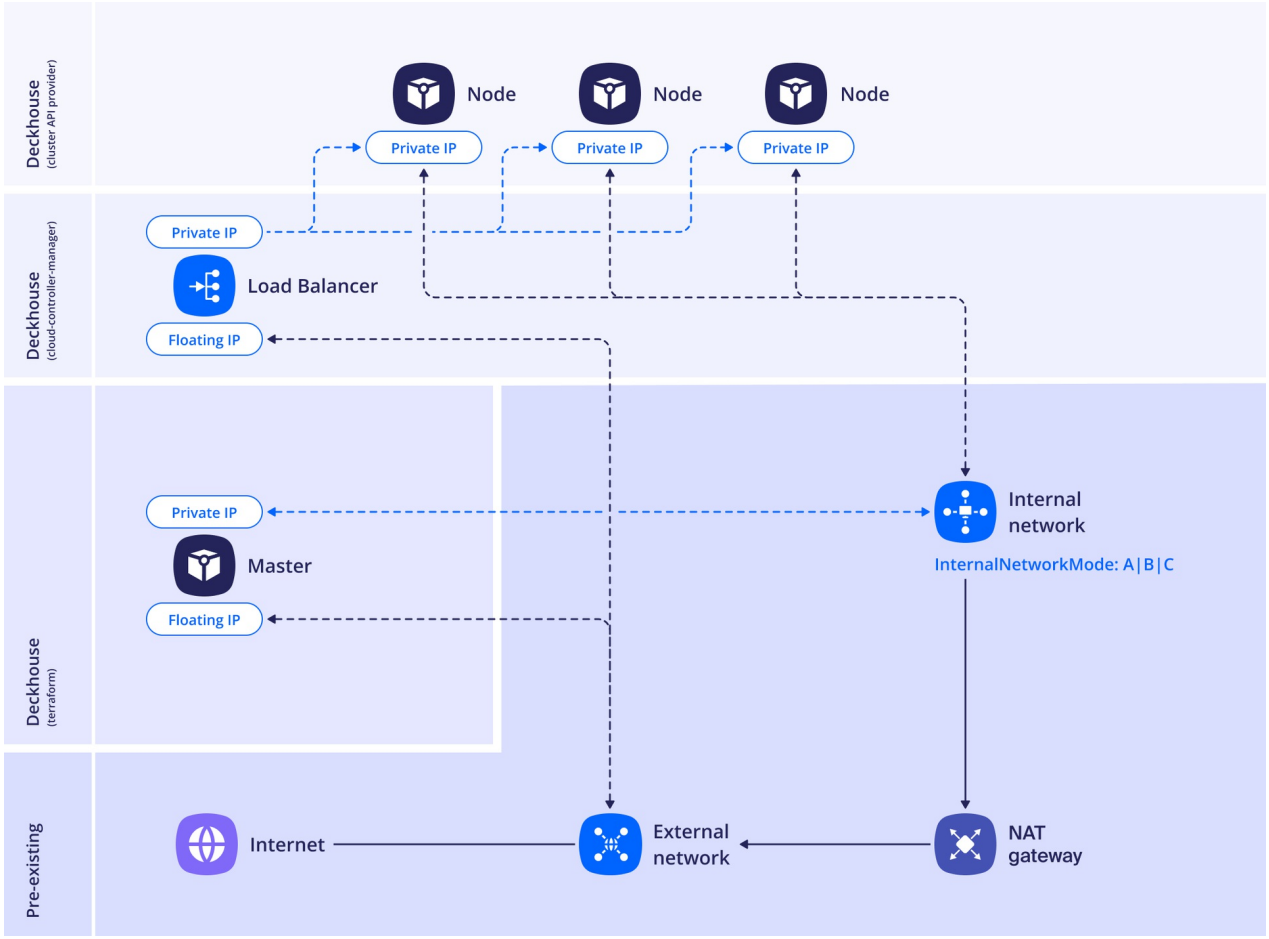
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, true by default.
  # Optional, VXLAN by default, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This layout can be useful if you need to merge a Kubernetes cluster with existing VMs.

> ⚠ This layout does not involve the management of SecurityGroups (it is assumed they were created beforehand). To configure security policies, you must explicitly specify both `additionalSecurityGroups` in the OpenStackClusterConfiguration for the masterNodeGroup and other nodeGroups, and `additionalSecurityGroups` when creating OpenStackInstanceClass in the cluster.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Required, all cluster nodes have to be in the same subnet.
  internalSubnetName: pivot-standard
  # Optional, DirectRoutingWithPortSecurityEnabled by default,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional. If set, it will be used for the load balancer default
  # configuration and ordering the master floating IP.
  externalNetworkName: ext-net
  # Optional, true by default.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Configuration

Integration with VK Cloud is performed using the OpenStackClusterConfiguration resource. It defines the configuration of the cloud cluster in VK Cloud and is used by the cloud provider when the cluster control plane is hosted in the cloud. The DKP module responsible for the integration is automatically configured based on the selected layout.

To modify the configuration in a running cluster, run the following command:

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

> **i** After changing node-related parameters, run the `dhctl converge` command to apply the changes.

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the VK Cloud cloud provider, the instance class is a custom resource called OpenStackInstanceClass, which contains the specific configuration of the VMs.

> **i** An OpenStack cluster is deployed in a single region (`provider.region`). Zones from other regions cannot be used. For details on configuring zones for CloudPermanent and CloudEphemeral nodes, see How do I create NodeGroups in availability zones? section.

> ⚠ When the module settings are changed, **existing Machine objects in the cluster are NOT recreated** (new Machine objects will use the updated parameters). The recreation only occurs if you modify the NodeGroup or the OpenStackInstanceClass parameters.

Configuration examples

The following are two simple configuration examples for the OpenStack cloud provider.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

List of required services

Below is the list of VK Cloud services required for DKP to operate in VK Cloud:

Service	API version
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* If you need to provision a LoadBalancer.

For the API endpoints and ports, refer to the official documentation.

LoadBalancer configuration

⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

It is recommended to use `loadbalancer.openstack.org/node-selector` to select only the nodes that should be used as targets for the corresponding LoadBalancer.

IngressNginxController example

In this example, the Ingress controller pods are scheduled on frontend nodes, while the `loadbalancer.openstack.org/node-selector` annotation limits the load balancer pool to the same nodes:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Node security policies and configuration

There are many reasons why you might want to restrict or allow incoming or outgoing traffic on cluster VMs. For example:

- Allow connections to cluster nodes from VMs in a different subnet.
- Allow access to specific ports on a static node for application traffic.
- Restrict access to external resources or other VMs in the cloud per security team requirements.

To manage this, you should use additional security groups. Only security groups that were pre-created in the cloud can be used.

Assigning additional security groups to static and master nodes

You can specify additional security groups either during cluster creation or in an existing cluster. In both cases, specify them in the OpenStackClusterConfiguration resource:

- **For master nodes**: Under the `masterNodeGroup` section using the `additionalSecurityGroups` field.
- **For static nodes**: Under the `nodeGroups` section of the corresponding node group configuration and also in the `additionalSecurityGroups` field.

The `additionalSecurityGroups` field is an array of strings representing the names of the security groups.

Assigning additional security groups to ephemeral nodes

To assign additional security groups to ephemeral nodes, specify the `additionalSecurityGroups` parameter in all relevant OpenStackInstanceClass resources used by these nodes.

Uploading an image to VK Cloud

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an openrc file that contains credentials for accessing the VK Cloud API.

   > **i** The method of obtaining the openrc file may vary depending on the VK Cloud provider. If your provider offers the standard interface for VK Cloud, follow this guide to download the file.

3. Alternatively, install the OpenStack CLI tools following the instructions.

   You can also run a container, mounting both the openrc file and the downloaded Ubuntu image:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize environment variables from the openrc file:

   ```
   source /openrc
   ```

5. List the available volume types:

   ```
   openstack volume type list
   ```

   Example output:

   ```
   +--------------------------------------+---------------+-----------+
   | ID                                   | Name          | Is Public |
   +--------------------------------------+---------------+-----------+
   | 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
   | bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
   | 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
   | eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
   | 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
   | a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
   | ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
   | a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
   +--------------------------------------+---------------+-----------+
   ```

6. Create an image and assign the volume type as a property (this is necessary if VK Cloud does not support local disks or they are unsuitable for workloads):

   ```
   openstack image create --private --disk-format qcow2 --container-format bare \
     --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
   ```

7. Ensure the image has been created:

   ```
   openstack image show ubuntu-18-04-cloud-amd64
   ```

   Example output:

```
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| Field            | Value
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| checksum         | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-
4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url':
u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap',
u'metadata': {}}]' |
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
```

Verifying security group support by the cloud provider

Run the `openstack security group list` command. If you don’t receive any errors, it means that security groups are supported.

Configuring online volume resize

When resizing a disk via the OpenStack API, Cinder does not pass updated size information to Nova. As a result, the disk size inside the guest OS remains unchanged.

To fix this, you must configure access to the Nova API in the `cinder.conf` file. For example:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

For more details, see the OpenStack-Ansible bug report.

Using rootDiskSize

Disks in VK Cloud

A node's disk can be either local or network-attached. In VK Cloud terminology:

- A local disk is called an ephemeral disk.
- A network-attached disk is called a persistent disk (Cinder storage).

The ephemeral disk is deleted along with the VM, while the persistent disk remains in the cloud even after the VM is deleted.

- For master nodes, network-attached disks are preferred so the nodes could migrate between hypervisors.
- For ephemeral nodes, local disks are preferred for cost efficiency. However, not all cloud providers support local disks. If not supported, you have to use network-attached disks for ephemeral nodes.

Local disk (ephemeral)	Network-attached disk (persistent)
Deleted with the VM	Remains in cloud and can be reused
Less expensive	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

rootDiskSize parameter

The OpenStackInstanceClass resource includes a `rootDiskSize` parameter. In VK Cloud, flavors may also define a disk size.

The resulting disk type depends on the combination of the following parameters:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` not set	*Disk size must be specified*. VM creation will fail without it.	Local disk with size from the flavor.
`rootDiskSize` set	Network-attached disk with size that equals `rootDiskSize`.	Both network-attached (`rootDiskSize`) and local (from flavor). Avoid this setup, as the cloud provider may charge for both.

> **i** In Selectel Cloud, when creating nodes with CloudEphemeral type outside zone A, you must pre-create a flavor with the target disk size. In this case, you do not need to set the `rootDiskSize` parameter.

Recommendations for master nodes and bastion with network-attached disks

- Use a flavor with zero disk size.
- Set `rootDiskSize` in the OpenStackInstanceClass resource.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the type will be taken from `volumeTypeMap`.

Recommendations for ephemeral nodes with local disks

- Use a flavor with a defined disk size.
- Do not use the `rootDiskSize` parameter in OpenStackInstanceClass.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the cloud provider's default disk type will be used.

Checking the disk size in a flavor

Run the following command:

```
openstack flavor show m1.medium-50g -C disk
```

Example output:

```
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

Overriding the cloud provider’s default disk type

If your cloud provider offers multiple volume types, you can specify the default type for an image by setting the `cinder_img_volume_type` property in the image metadata. Any VM created from that image will use the specified volume type.

You can also create a new VK Cloud image and upload it.

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Connection and authorization in Selectel Cloud

To manage resources in Selectel using Deckhouse Kubernetes Platform, you must connect to the Selectel API.

The list of Selectel API services required for deployment is available in the Configuration section.

The user credentials required to access the Selectel API are provided in the OpenStack RC file (openrc file).

Instructions for downloading the openrc file via the standard web interface of Selectel and using it are available in the Selectel documentation.

If you’re using a cloud provider’s implementation of the Selectel API, the process for obtaining the openrc file may differ.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Layouts and configuration in Selectel Cloud

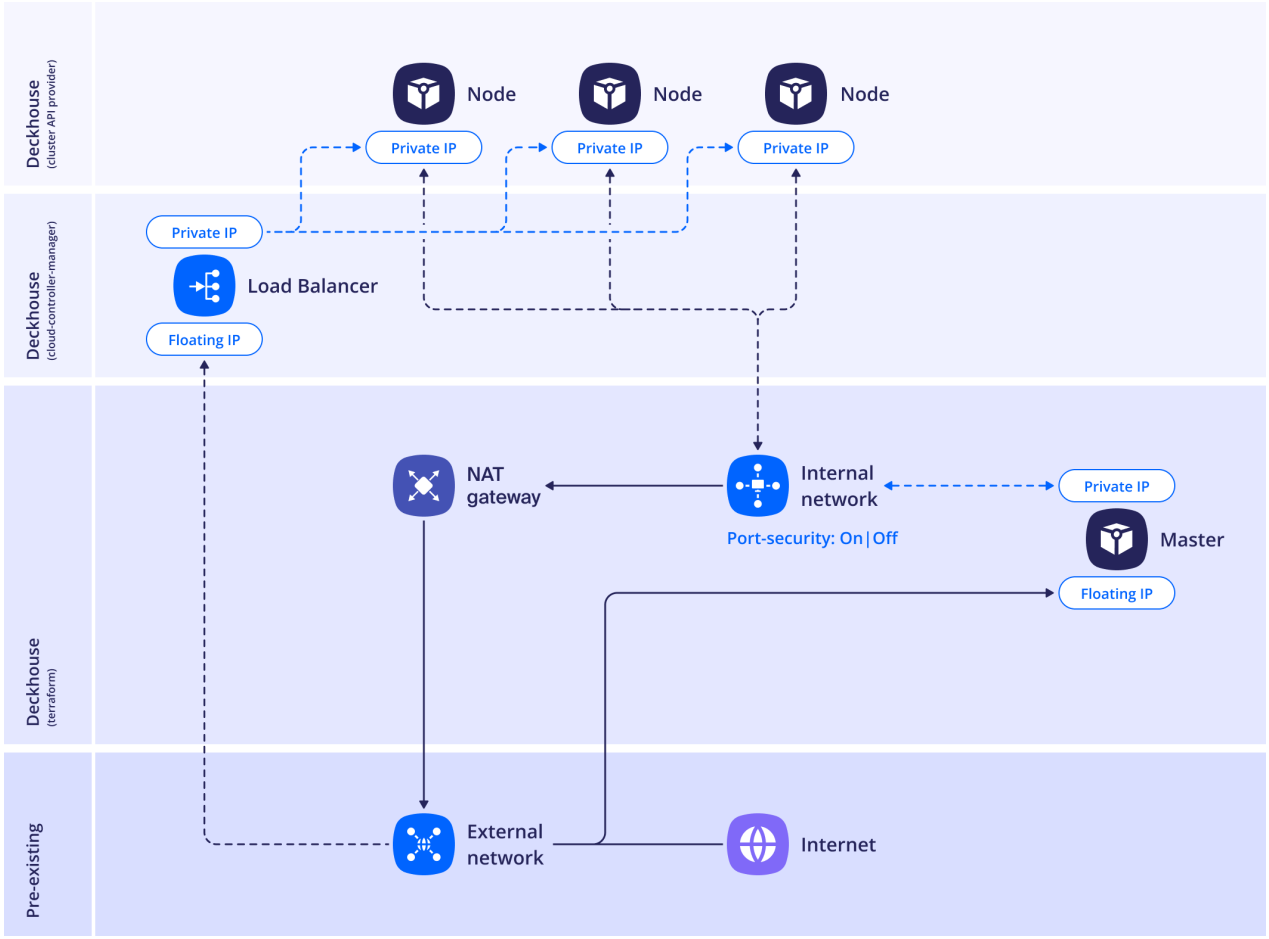
Layouts

This section describes the possible node placement layouts in Selectel infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. The floating IP is assigned to the master node.

⚠ If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port `6443`. To avoid this, we recommend using the SimpleWithInternalNetwork or Standard layout with a bastion host.



Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                 # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, true by default.
  externalNetworkName: shared                # Required.
  bastion:
    zone: ru2-b                             # Optional.
    volumeType: fast-ru-2b                   # Optional.
    instanceClass:
      flavorName: m1.large                   # Required.
      imageName: ubuntu-20-04-cloud-amd64      # Required.
      rootDiskSize: 50                      # Optional, 50 GB by default.
      additionalTags:
        severity: critical                   # Optional.
        environment: production              # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
```

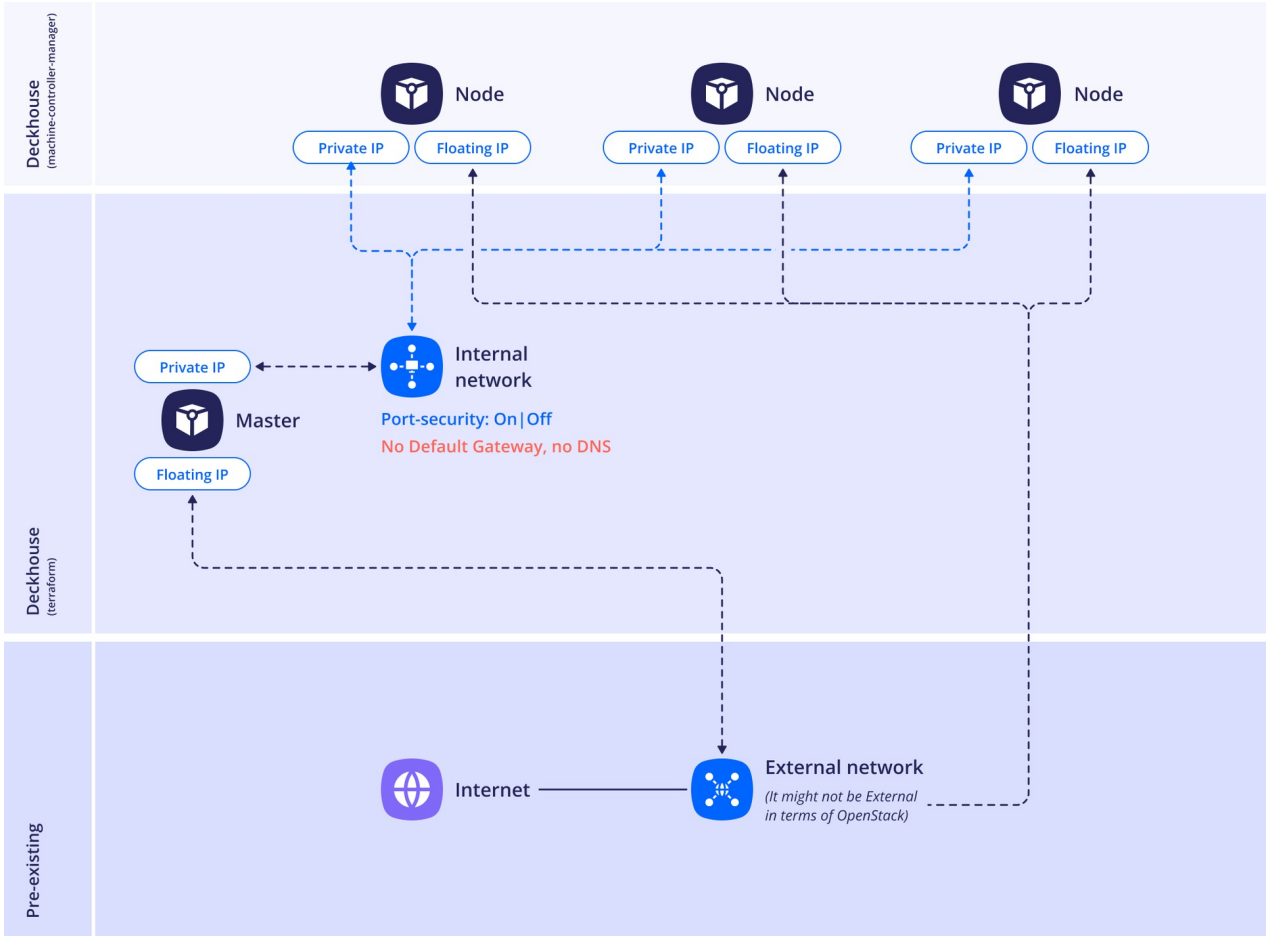
```
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                  # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default. Determines if configuration drive is
    # required during vm bootstrap process. It's needed if there
    # is no DHCP in the network that is used as a default gateway.
    configDrive: false
    # Required, the gateway of this network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                   # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

⚠

- This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.
- Using this strategy, make sure to explicitly specify the name of the internal network in `additionalNetworks` when creating an OpenStackInstanceClass in the cluster.



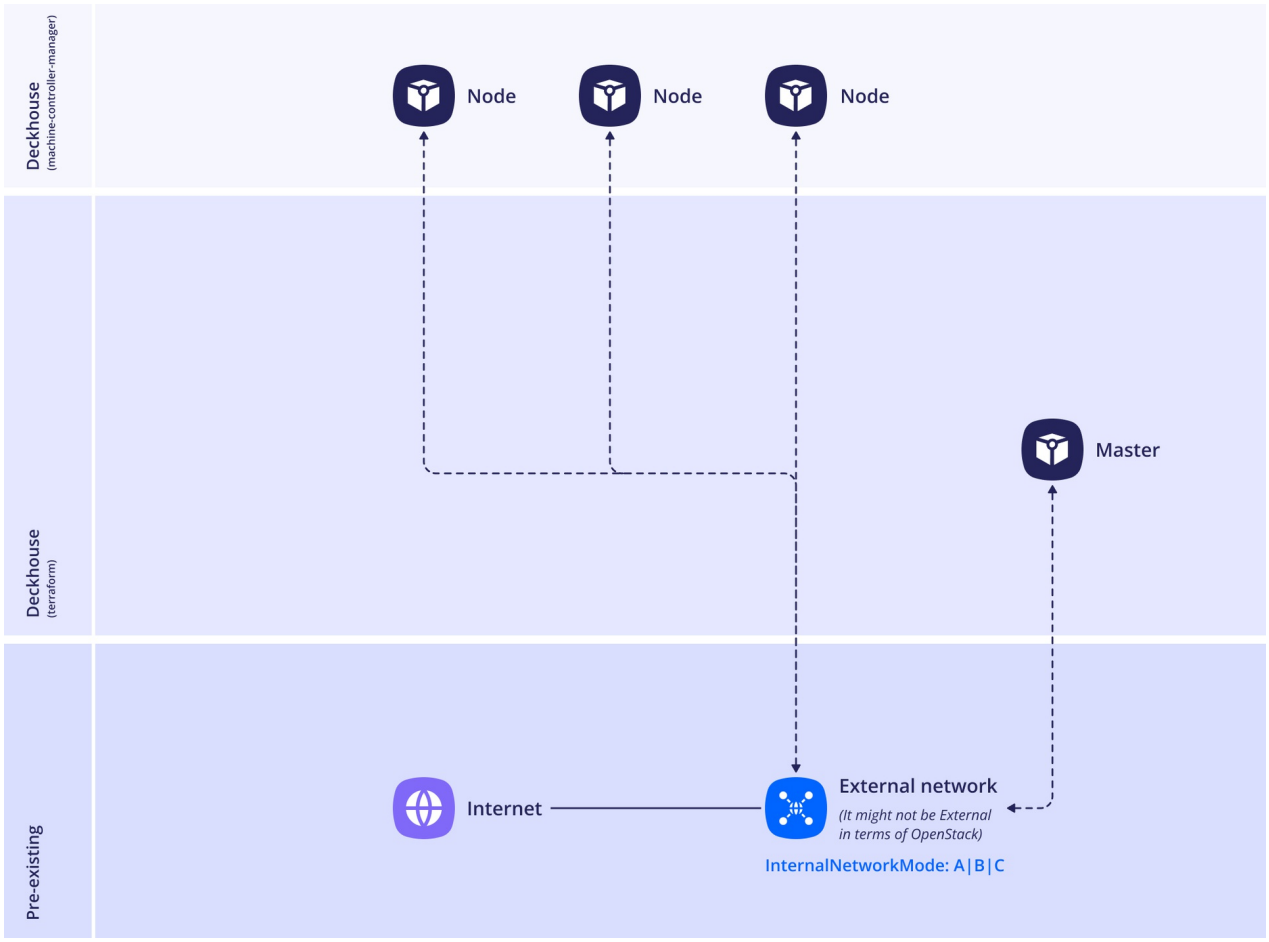
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, defines whether DHCP is enabled in specified external
  # network (true by default).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, true by default.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy can be useful if you need to merge a Kubernetes cluster with existing VMs.

> ⚠ This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.



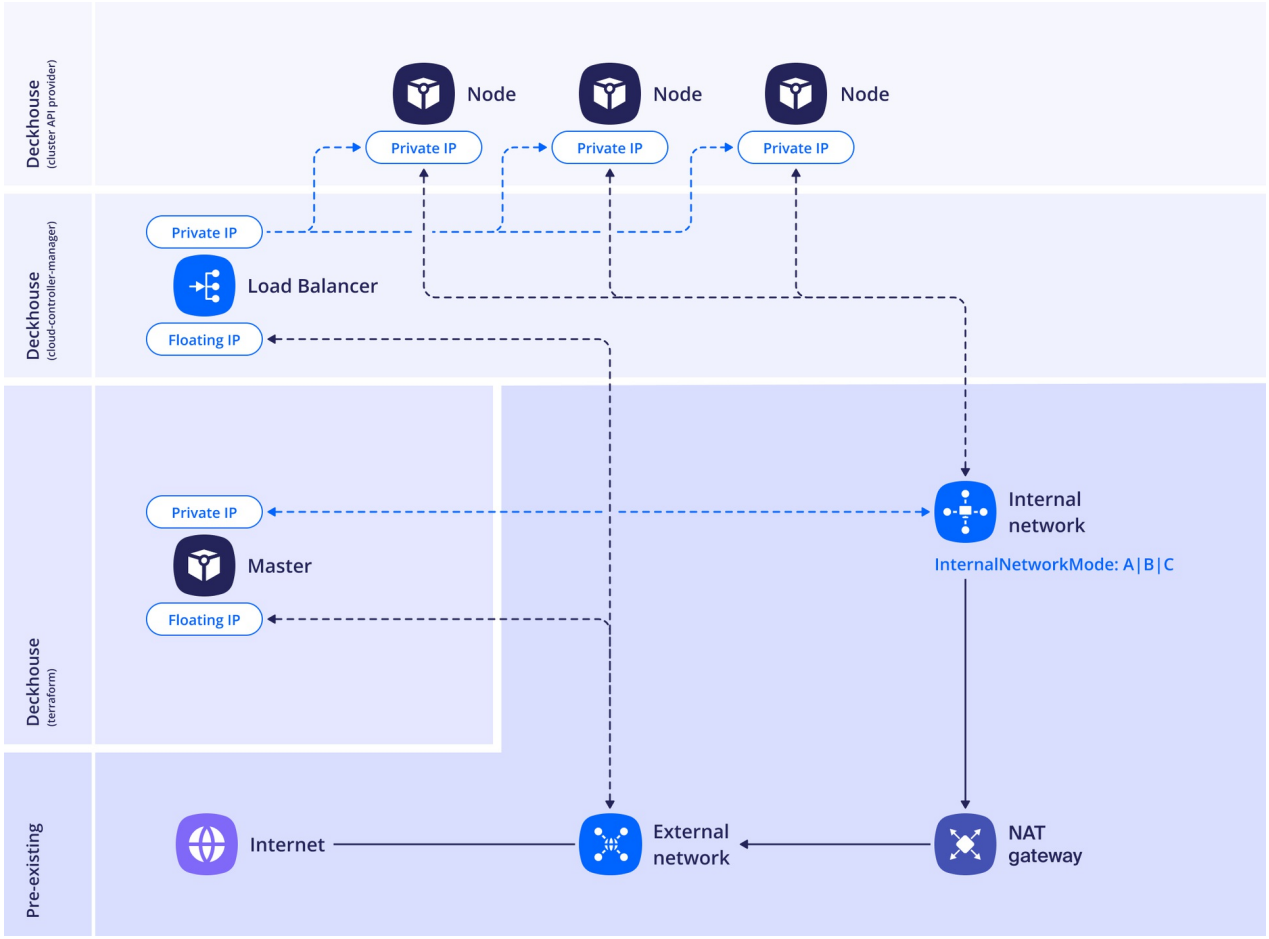
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, true by default.
  # Optional, VXLAN by default, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This layout can be useful if you need to merge a Kubernetes cluster with existing VMs.

> ⚠ This layout does not involve the management of SecurityGroups (it is assumed they were created beforehand). To configure security policies, you must explicitly specify both `additionalSecurityGroups` in the OpenStackClusterConfiguration for the masterNodeGroup and other nodeGroups, and `additionalSecurityGroups` when creating OpenStackInstanceClass in the cluster.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Required, all cluster nodes have to be in the same subnet.
  internalSubnetName: pivot-standard
  # Optional, DirectRoutingWithPortSecurityEnabled by default,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional. If set, it will be used for the load balancer default
  # configuration and ordering the master floating IP.
  externalNetworkName: ext-net
  # Optional, true by default.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Configuration

Integration with Selectel is performed using the OpenStackClusterConfiguration resource. It defines the configuration of the cloud cluster in Selectel and is used by the cloud provider when the cluster control plane is hosted in the cloud. The DKP module responsible for the integration is automatically configured based on the selected layout.

To modify the configuration in a running cluster, run the following command:

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

> **i** After changing node-related parameters, run the `dhctl converge` command to apply the changes.

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the Selectel cloud provider, the instance class is a custom resource called OpenStackInstanceClass, which contains the specific configuration of the VMs.

> **i** An OpenStack cluster is deployed in a single region (`provider.region`). Zones from other regions cannot be used. For details on configuring zones for CloudPermanent and CloudEphemeral nodes, see How do I create NodeGroups in availability zones? section.

> ⚠ When the module settings are changed, **existing Machine objects in the cluster are NOT recreated** (new Machine objects will use the updated parameters). The recreation only occurs if you modify the NodeGroup or the OpenStackInstanceClass parameters.

Configuration examples

The following are two simple configuration examples for the OpenStack cloud provider.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

List of required services

Below is the list of Selectel services required for DKP to operate in Selectel:

Service	API version
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* If you need to provision a LoadBalancer.

LoadBalancer configuration

⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

It is recommended to use `loadbalancer.openstack.org/node-selector` to select only the nodes that should be used as targets for the corresponding LoadBalancer.

IngressNginxController example

In this example, the Ingress controller pods are scheduled on frontend nodes, while the `loadbalancer.openstack.org/node-selector` annotation limits the load balancer pool to the same nodes:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Node security policies and configuration

There are many reasons why you might want to restrict or allow incoming or outgoing traffic on cluster VMs. For example:

- Allow connections to cluster nodes from VMs in a different subnet.
- Allow access to specific ports on a static node for application traffic.
- Restrict access to external resources or other VMs in the cloud per security team requirements.

To manage this, you should use additional security groups. Only security groups that were pre-created in the cloud can be used.

Assigning additional security groups to static and master nodes

You can specify additional security groups either during cluster creation or in an existing cluster. In both cases, specify them in the OpenStackClusterConfiguration resource:

- **For master nodes**: Under the `masterNodeGroup` section using the `additionalSecurityGroups` field.
- **For static nodes**: Under the `nodeGroups` section of the corresponding node group configuration and also in the `additionalSecurityGroups` field.

The `additionalSecurityGroups` field is an array of strings representing the names of the security groups.

Assigning additional security groups to ephemeral nodes

To assign additional security groups to ephemeral nodes, specify the `additionalSecurityGroups` parameter in all

relevant OpenStackInstanceClass resources used by these nodes.

Uploading an image to Selectel

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an openrc file that contains credentials for accessing the Selectel API.

   > **i** The method of obtaining the openrc file may vary depending on the Selectel provider. If your provider offers the standard interface for Selectel, follow this guide to download the file.

3. Alternatively, install the OpenStack CLI tools following the instructions.

   You can also run a container, mounting both the openrc file and the downloaded Ubuntu image:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize environment variables from the openrc file:

   ```
   source /openrc
   ```

5. List the available volume types:

   ```
   openstack volume type list
   ```

   Example output:

   ```
   +--------------------------------------+---------------+-----------+
   | ID                                   | Name          | Is Public |
   +--------------------------------------+---------------+-----------+
   | 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
   | bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
   | 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
   | eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
   | 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
   | a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
   | ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
   | a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
   +--------------------------------------+---------------+-----------+
   ```

6. Create an image and assign the volume type as a property (this is necessary if Selectel does not support local disks or they are unsuitable for workloads):

   ```
   openstack image create --private --disk-format qcow2 --container-format bare \
     --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
   ```

7. Ensure the image has been created:

   ```
   openstack image show ubuntu-18-04-cloud-amd64
   ```

   Example output:

```
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| Field            | Value
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| checksum         | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-
4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url':
u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap',
u'metadata': {}}]' |
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
```

Verifying security group support by the cloud provider

Run the `openstack security group list` command. If you don’t receive any errors, it means that security groups are supported.

Configuring online volume resize

When resizing a disk via the OpenStack API, Cinder does not pass updated size information to Nova. As a result, the disk size inside the guest OS remains unchanged.

To fix this, you must configure access to the Nova API in the `cinder.conf` file. For example:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

For more details, see the OpenStack-Ansible bug report.

Using rootDiskSize

Disks in Selectel

A node’s disk can be either local or network-attached. In Selectel terminology:

- A local disk is called an ephemeral disk.
- A network-attached disk is called a persistent disk (Cinder storage).

The ephemeral disk is deleted along with the VM, while the persistent disk remains in the cloud even after the VM is deleted.

- For master nodes, network-attached disks are preferred so the nodes could migrate between hypervisors.
- For ephemeral nodes, local disks are preferred for cost efficiency. However, not all cloud providers support local disks. If not supported, you have to use network-attached disks for ephemeral nodes.

Local disk (ephemeral)	Network-attached disk (persistent)
Deleted with the VM	Remains in cloud and can be reused
Less expensive	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

rootDiskSize parameter

The OpenStackInstanceClass resource includes a `rootDiskSize` parameter. In Selectel, flavors may also define a disk size.

The resulting disk type depends on the combination of the following parameters:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` not set	*Disk size must be specified*. VM creation will fail without it.	Local disk with size from the flavor.
`rootDiskSize` set	Network-attached disk with size that equals `rootDiskSize`.	Both network-attached (`rootDiskSize`) and local (from flavor). Avoid this setup, as the cloud provider may charge for both.

Recommendations for master nodes and bastion with network-attached disks

- Use a flavor with zero disk size.
- Set `rootDiskSize` in the OpenStackInstanceClass resource.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the type will be taken from `volumeTypeMap`.

Recommendations for ephemeral nodes with local disks

- Use a flavor with a defined disk size.
- Do not use the `rootDiskSize` parameter in OpenStackInstanceClass.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the cloud provider’s default disk type will be used.

Checking the disk size in a flavor

Run the following command:

```
openstack flavor show m1.medium-50g -C disk
```

Example output:

```
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

Overriding the cloud provider’s default disk type

If your cloud provider offers multiple volume types, you can specify the default type for an image by setting the `cinder_img_volume_type` property in the image metadata. Any VM created from that image will use the specified volume type.

You can also create a new Selectel image and upload it.

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Offline disk resizing

If you encounter the following error during volume resizing, reduce the number of replicas in the affected StatefulSet to 0, wait for the volume size to update, and then scale the replicas back to the original value.

```
Warning  VolumeResizeFailed     5s (x11 over 41s)  external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-
9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

Connection and authorization in OVH Cloud

To manage resources in OVHcloud using Deckhouse Kubernetes Platform, you must connect to the OVHcloud API.

The list of OVHcloud API services required for deployment is available in the Configuration section.

The user credentials required to access the OVHcloud API are provided in the OpenStack RC file (openrc file).

Instructions for downloading the openrc file via the standard web interface of OVHcloud and using it are available in the OVHcloud documentation.

If you’re using a cloud provider’s implementation of the OVHcloud API, the process for obtaining the openrc file may differ.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Layouts and configuration in OVH Cloud

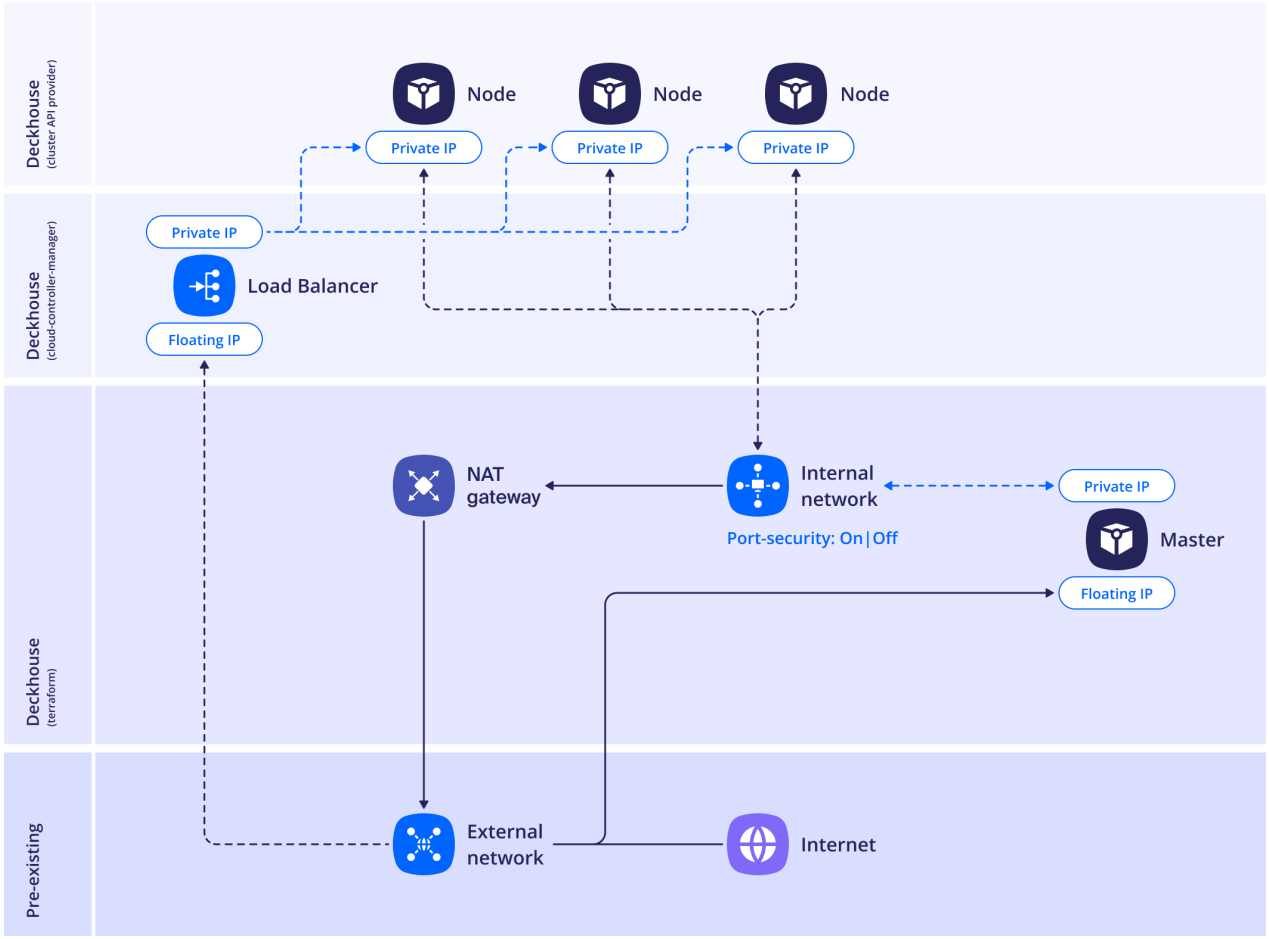
Layouts

This section describes the possible node placement layouts in OVHcloud infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. The floating IP is assigned to the master node.

⚠ If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port `6443`. To avoid this, we recommend using the SimpleWithInternalNetwork or Standard layout with a bastion host.



Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                  # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, true by default.
  externalNetworkName: shared                 # Required.
  bastion:
    zone: ru2-b                             # Optional.
    volumeType: fast-ru-2b                    # Optional.
    instanceClass:
      flavorName: m1.large                    # Required.
      imageName: ubuntu-20-04-cloud-amd64     # Required.
      rootDiskSize: 50                      # Optional, 50 GB by default.
      additionalTags:
        severity: critical                  # Optional.
        environment: production             # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
```

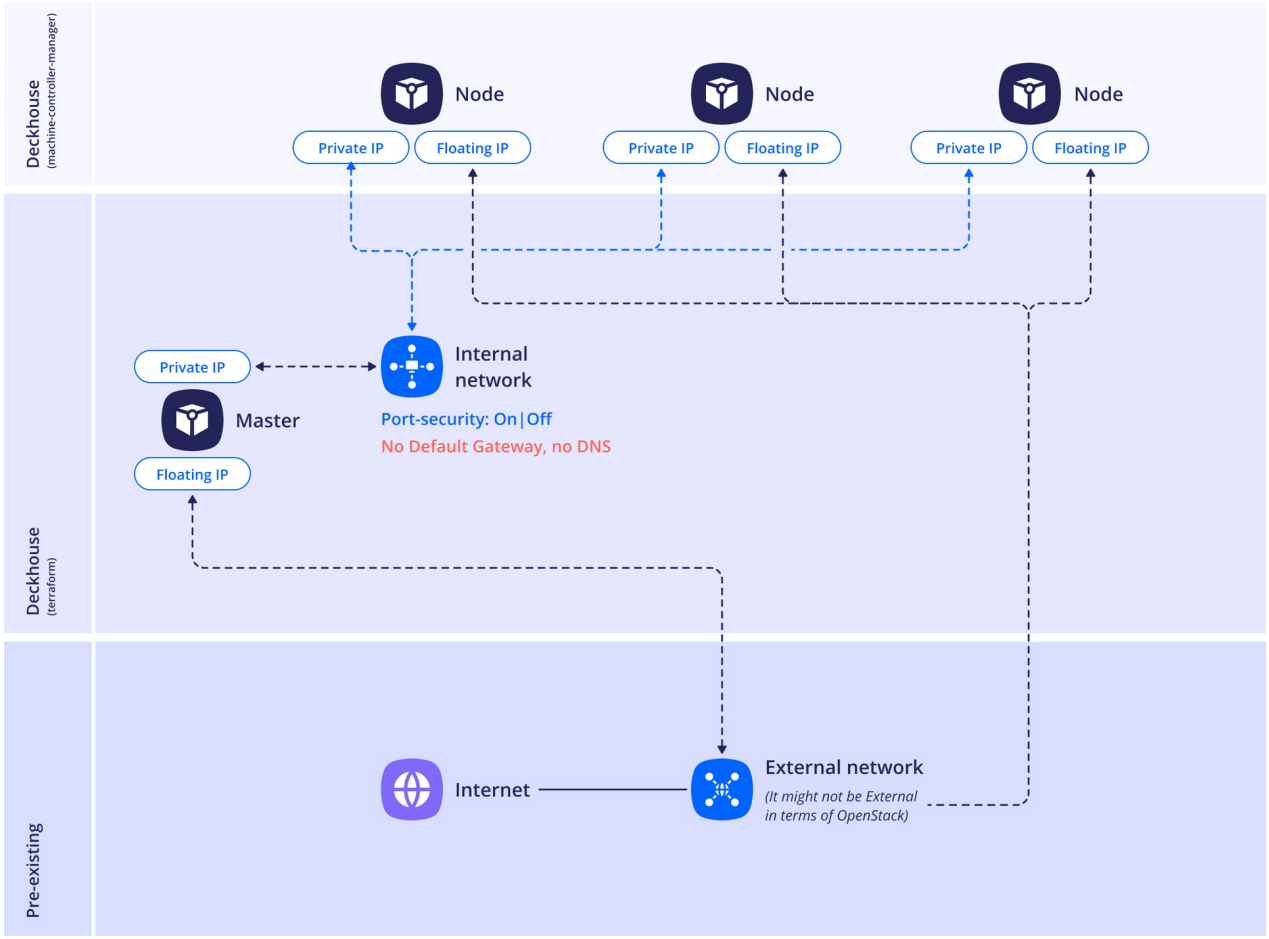
```
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                  # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default. Determines if configuration drive is
    # required during vm bootstrap process. It's needed if there
    # is no DHCP in the network that is used as a default gateway.
    configDrive: false
    # Required, the gateway of this network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                   # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

⚠

- This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.
- Using this strategy, make sure to explicitly specify the name of the internal network in `additionalNetworks` when creating an OpenStackInstanceClass in the cluster.



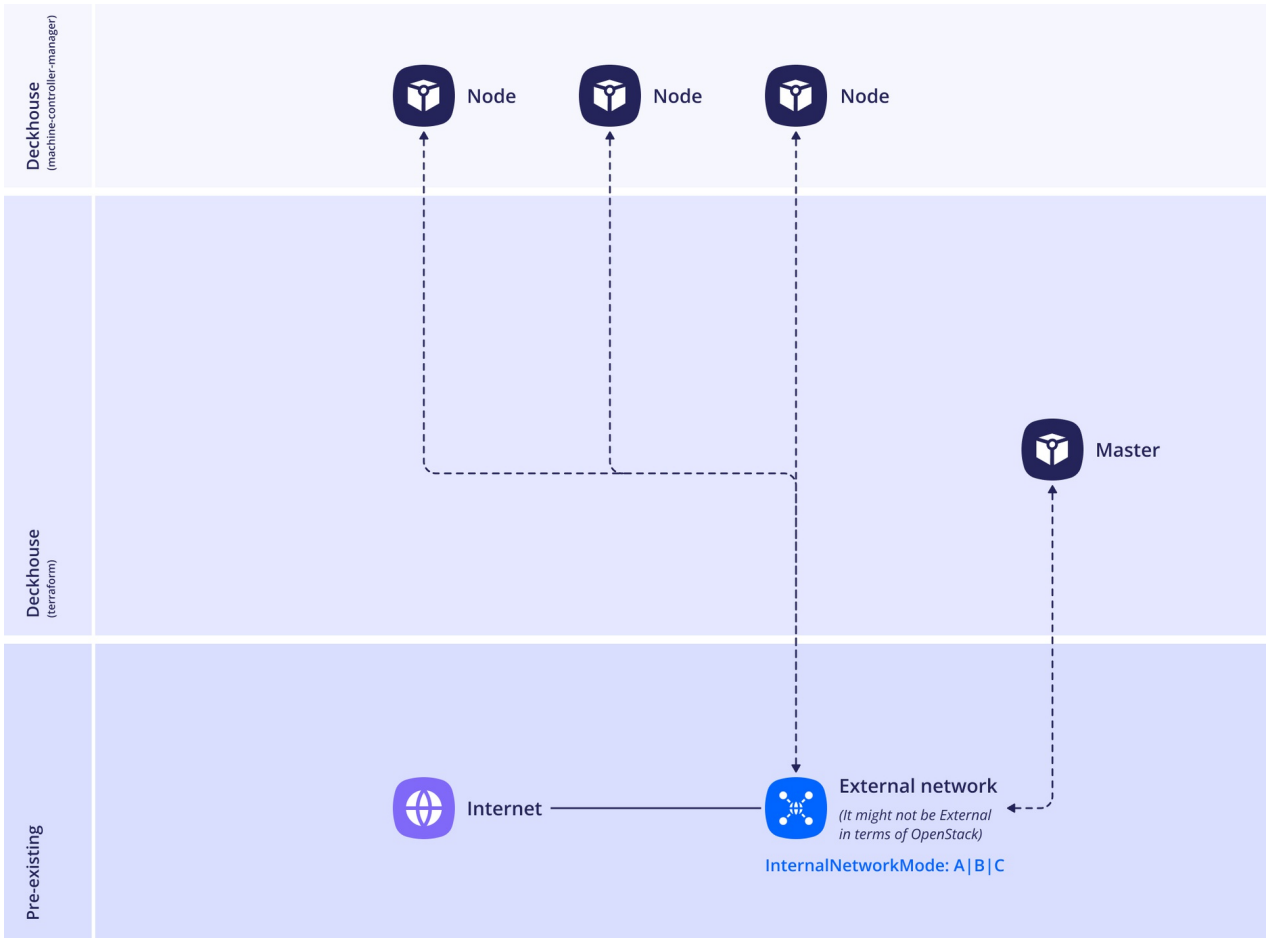
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, defines whether DHCP is enabled in specified external
  # network (true by default).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, true by default.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy can be useful if you need to merge a Kubernetes cluster with existing VMs.

⚠ This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.



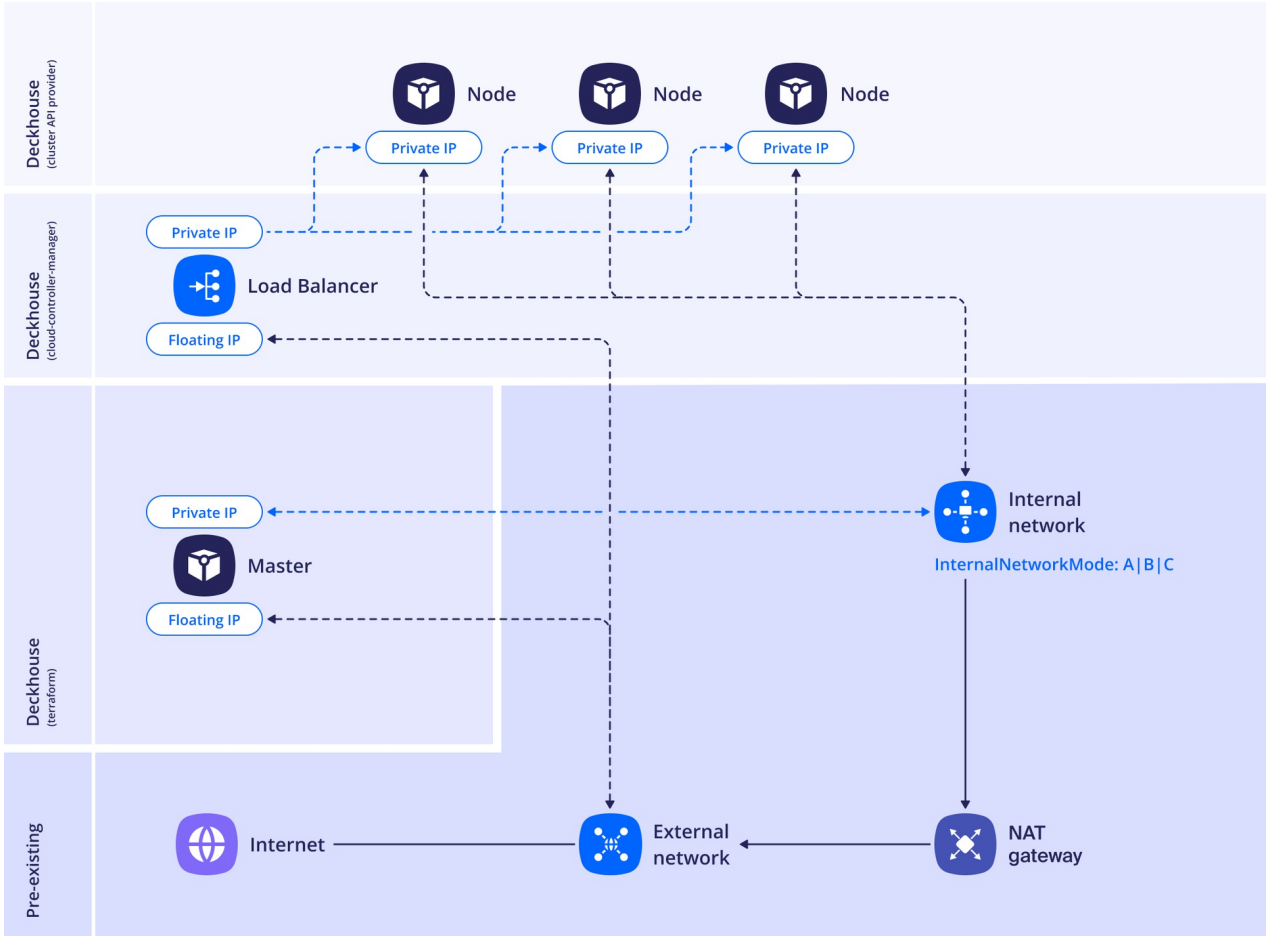
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, true by default.
  # Optional, VXLAN by default, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This layout can be useful if you need to merge a Kubernetes cluster with existing VMs.

> ⚠ This layout does not involve the management of SecurityGroups (it is assumed they were created beforehand). To configure security policies, you must explicitly specify both `additionalSecurityGroups` in the OpenStackClusterConfiguration for the masterNodeGroup and other nodeGroups, and `additionalSecurityGroups` when creating OpenStackInstanceClass in the cluster.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Required, all cluster nodes have to be in the same subnet.
  internalSubnetName: pivot-standard
  # Optional, DirectRoutingWithPortSecurityEnabled by default,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional. If set, it will be used for the load balancer default
  # configuration and ordering the master floating IP.
  externalNetworkName: ext-net
  # Optional, true by default.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Configuration

Integration with OVHcloud is performed using the OpenStackClusterConfiguration resource. It defines the configuration of the cloud cluster in OVHcloud and is used by the cloud provider when the cluster control plane is hosted in the cloud. The DKP module responsible for the integration is automatically configured based on the selected layout.

To modify the configuration in a running cluster, run the following command:

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

> **i** After changing node-related parameters, run the `dhctl converge` command to apply the changes.

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the OVHcloud cloud provider, the instance class is a custom resource called OpenStackInstanceClass, which contains the specific configuration of the VMs.

> **i** An OpenStack cluster is deployed in a single region (`provider.region`). Zones from other regions cannot be used. For details on configuring zones for CloudPermanent and CloudEphemeral nodes, see How do I create NodeGroups in availability zones? section.

> ⚠ When the module settings are changed, **existing Machine objects in the cluster are NOT recreated** (new Machine objects will use the updated parameters). The recreation only occurs if you modify the NodeGroup or the OpenStackInstanceClass parameters.

Configuration examples

The following are two simple configuration examples for the OpenStack cloud provider.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

List of required services

Below is the list of OVHcloud services required for DKP to operate in OVHcloud:

Service	API version
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* If you need to provision a LoadBalancer.

LoadBalancer configuration

⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

It is recommended to use `loadbalancer.openstack.org/node-selector` to select only the nodes that should be used as targets for the corresponding LoadBalancer.

IngressNginxController example

In this example, the Ingress controller pods are scheduled on frontend nodes, while the `loadbalancer.openstack.org/node-selector` annotation limits the load balancer pool to the same nodes:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Node security policies and configuration

There are many reasons why you might want to restrict or allow incoming or outgoing traffic on cluster VMs. For example:

- Allow connections to cluster nodes from VMs in a different subnet.
- Allow access to specific ports on a static node for application traffic.
- Restrict access to external resources or other VMs in the cloud per security team requirements.

To manage this, you should use additional security groups. Only security groups that were pre-created in the cloud can be used.

Assigning additional security groups to static and master nodes

You can specify additional security groups either during cluster creation or in an existing cluster. In both cases, specify them in the OpenStackClusterConfiguration resource:

- **For master nodes**: Under the `masterNodeGroup` section using the `additionalSecurityGroups` field.
- **For static nodes**: Under the `nodeGroups` section of the corresponding node group configuration and also in the `additionalSecurityGroups` field.

The `additionalSecurityGroups` field is an array of strings representing the names of the security groups.

Assigning additional security groups to ephemeral nodes

To assign additional security groups to ephemeral nodes, specify the `additionalSecurityGroups` parameter in all

relevant OpenStackInstanceClass resources used by these nodes.

Uploading an image to OVHcloud

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an openrc file that contains credentials for accessing the OVHcloud API.

   > **i** The method of obtaining the openrc file may vary depending on the OVHcloud provider. If your provider offers the standard interface for OVHcloud, follow this guide to download the file.

3. Alternatively, install the OpenStack CLI tools following the instructions.

   You can also run a container, mounting both the openrc file and the downloaded Ubuntu image:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize environment variables from the openrc file:

   ```
   source /openrc
   ```

5. List the available volume types:

   ```
   openstack volume type list
   ```

   Example output:

   ```
   +--------------------------------------+---------------+-----------+
   | ID                                   | Name          | Is Public |
   +--------------------------------------+---------------+-----------+
   | 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
   | bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
   | 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
   | eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
   | 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
   | a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
   | ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
   | a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
   +--------------------------------------+---------------+-----------+
   ```

6. Create an image and assign the volume type as a property (this is necessary if OVHcloud does not support local disks or they are unsuitable for workloads):

   ```
   openstack image create --private --disk-format qcow2 --container-format bare \
     --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
   ```

7. Ensure the image has been created:

   ```
   openstack image show ubuntu-18-04-cloud-amd64
   ```

   Example output:

```
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| Field            | Value
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| checksum         | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-
4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url':
u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap',
u'metadata': {}}]' |
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
```

Verifying security group support by the cloud provider

Run the `openstack security group list` command. If you don’t receive any errors, it means that security groups are supported.

Configuring online volume resize

When resizing a disk via the OpenStack API, Cinder does not pass updated size information to Nova. As a result, the disk size inside the guest OS remains unchanged.

To fix this, you must configure access to the Nova API in the `cinder.conf` file. For example:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

For more details, see the OpenStack-Ansible bug report.

Using rootDiskSize

Disks in OVHcloud

A node's disk can be either local or network-attached. In OVHcloud terminology:

- A local disk is called an ephemeral disk.
- A network-attached disk is called a persistent disk (Cinder storage).

The ephemeral disk is deleted along with the VM, while the persistent disk remains in the cloud even after the VM is deleted.

- For master nodes, network-attached disks are preferred so the nodes could migrate between hypervisors.
- For ephemeral nodes, local disks are preferred for cost efficiency. However, not all cloud providers support local disks. If not supported, you have to use network-attached disks for ephemeral nodes.

Local disk (ephemeral)	Network-attached disk (persistent)
Deleted with the VM	Remains in cloud and can be reused
Less expensive	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

rootDiskSize parameter

The OpenStackInstanceClass resource includes a `rootDiskSize` parameter. In OVHcloud, flavors may also define a disk size.

The resulting disk type depends on the combination of the following parameters:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` not set	*Disk size must be specified*. VM creation will fail without it.	Local disk with size from the flavor.
`rootDiskSize` set	Network-attached disk with size that equals `rootDiskSize`.	Both network-attached (`rootDiskSize`) and local (from flavor). Avoid this setup, as the cloud provider may charge for both.

> **i** In Selectel Cloud, when creating nodes with CloudEphemeral type outside zone A, you must pre-create a flavor with the target disk size. In this case, you do not need to set the `rootDiskSize` parameter.

Recommendations for master nodes and bastion with network-attached disks

- Use a flavor with zero disk size.
- Set `rootDiskSize` in the OpenStackInstanceClass resource.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the type will be taken from `volumeTypeMap`.

Recommendations for ephemeral nodes with local disks

- Use a flavor with a defined disk size.
- Do not use the `rootDiskSize` parameter in OpenStackInstanceClass.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the cloud provider's default disk type will be used.

Checking the disk size in a flavor

Run the following command:

```
openstack flavor show m1.medium-50g -C disk
```

Example output:

```
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

Overriding the cloud provider’s default disk type

If your cloud provider offers multiple volume types, you can specify the default type for an image by setting the `cinder_img_volume_type` property in the image metadata. Any VM created from that image will use the specified volume type.

You can also create a new OVHcloud image and upload it.

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Offline disk resizing

If you encounter the following error during volume resizing, reduce the number of replicas in the affected StatefulSet to 0, wait for the volume size to update, and then scale the replicas back to the original value.

```
Warning  VolumeResizeFailed     5s (x11 over 41s)  external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-
9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

Connection and authorization in OpenStack

To manage resources in OpenStack using Deckhouse Kubernetes Platform, you must connect to the OpenStack API.

The list of OpenStack API services required for deployment is available in the Configuration section.

The user credentials required to access the OpenStack API are provided in the OpenStack RC file (openrc file).

Instructions for downloading the openrc file via the standard web interface of OpenStack and using it are available in the OpenStack documentation.

If you’re using a cloud provider’s implementation of the OpenStack API, the process for obtaining the openrc file may differ.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Layouts and configuration in OpenStack

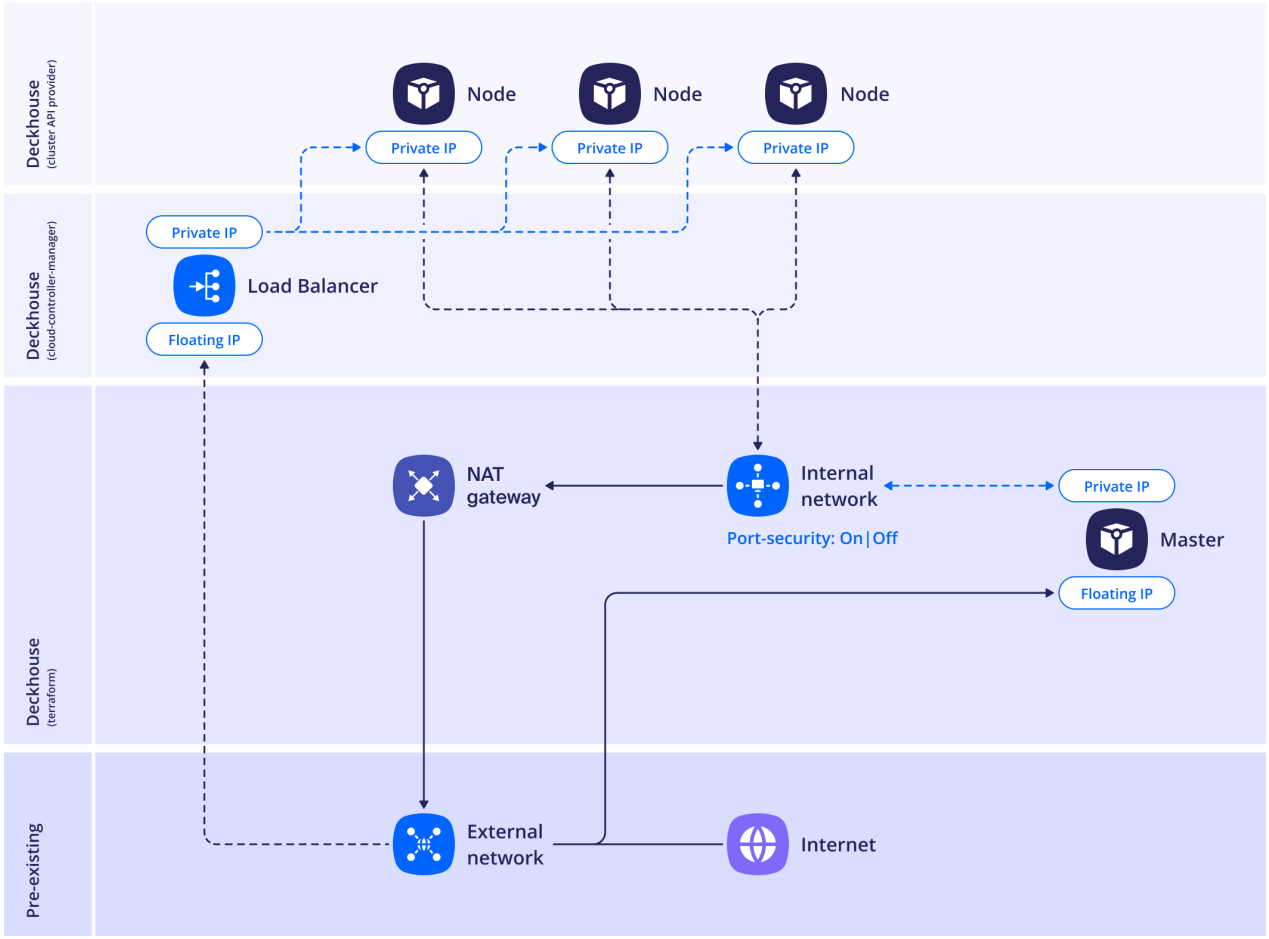
Layouts

This section describes the possible node placement layouts in OpenStack infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. The floating IP is assigned to the master node.

⚠ If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port `6443`. To avoid this, we recommend using the SimpleWithInternalNetwork or Standard layout with a bastion host.



Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                 # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, true by default.
  externalNetworkName: shared                # Required.
  bastion:
    zone: ru2-b                             # Optional.
    volumeType: fast-ru-2b                   # Optional.
    instanceClass:
      flavorName: m1.large                   # Required.
      imageName: ubuntu-20-04-cloud-amd64      # Required.
      rootDiskSize: 50                      # Optional, 50 GB by default.
      additionalTags:
        severity: critical                   # Optional.
        environment: production              # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
```

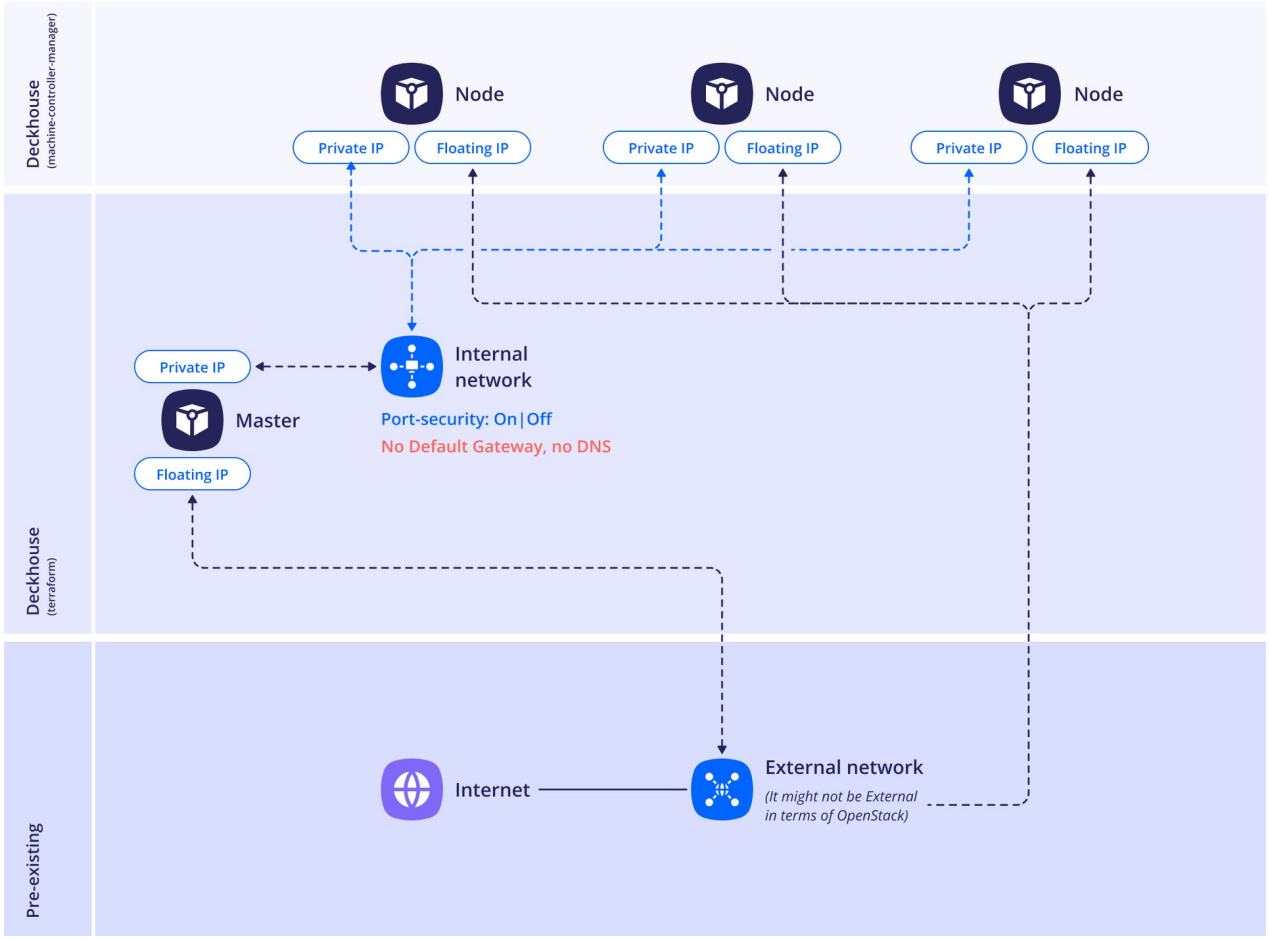
```
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                  # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default. Determines if configuration drive is
    # required during vm bootstrap process. It's needed if there
    # is no DHCP in the network that is used as a default gateway.
    configDrive: false
    # Required, the gateway of this network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                   # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

⚠

- This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.
- Using this strategy, make sure to explicitly specify the name of the internal network in `additionalNetworks` when creating an OpenStackInstanceClass in the cluster.



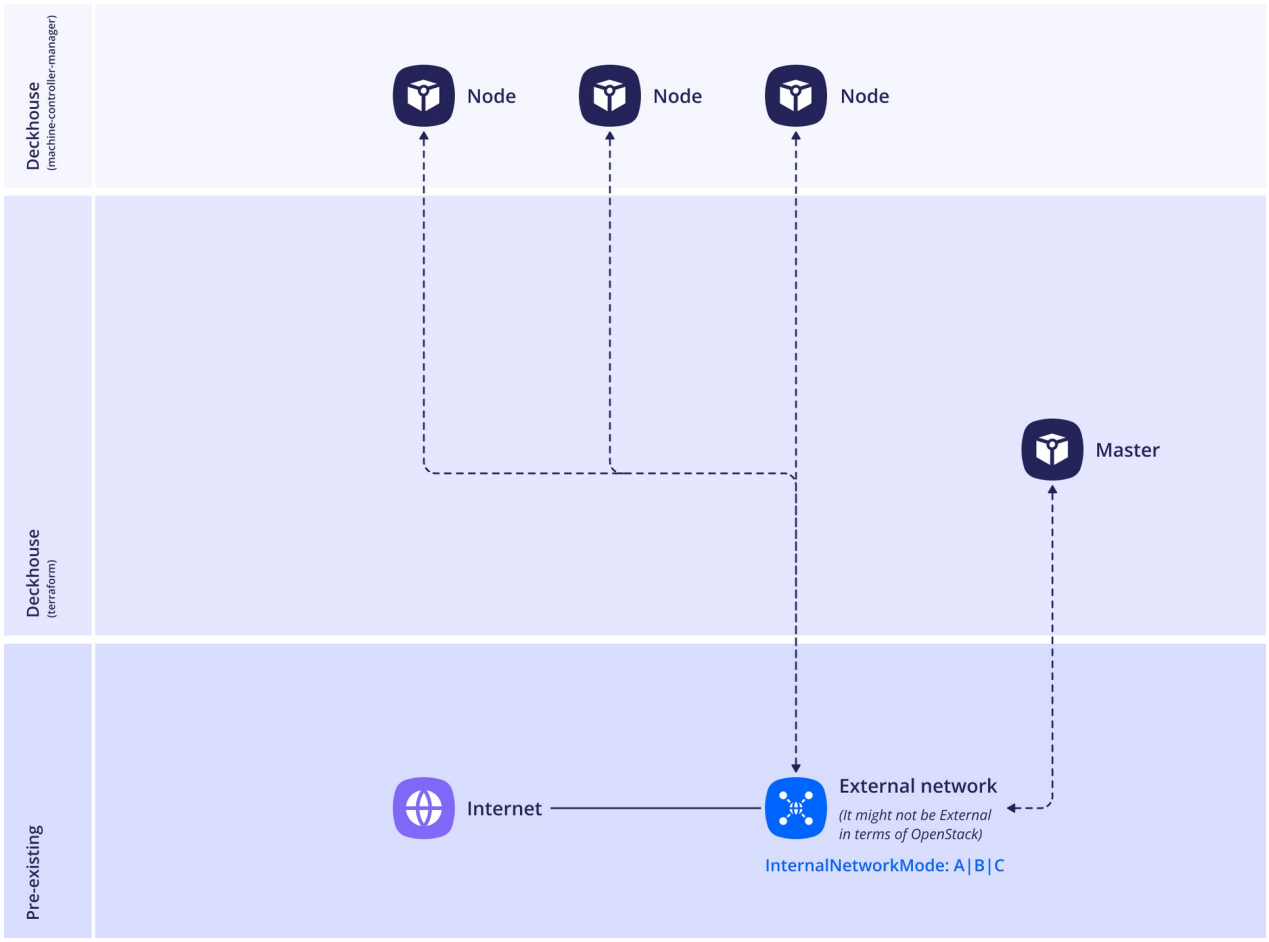
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, defines whether DHCP is enabled in specified external
  # network (true by default).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, true by default.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy can be useful if you need to merge a Kubernetes cluster with existing VMs.

> ⚠ This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.



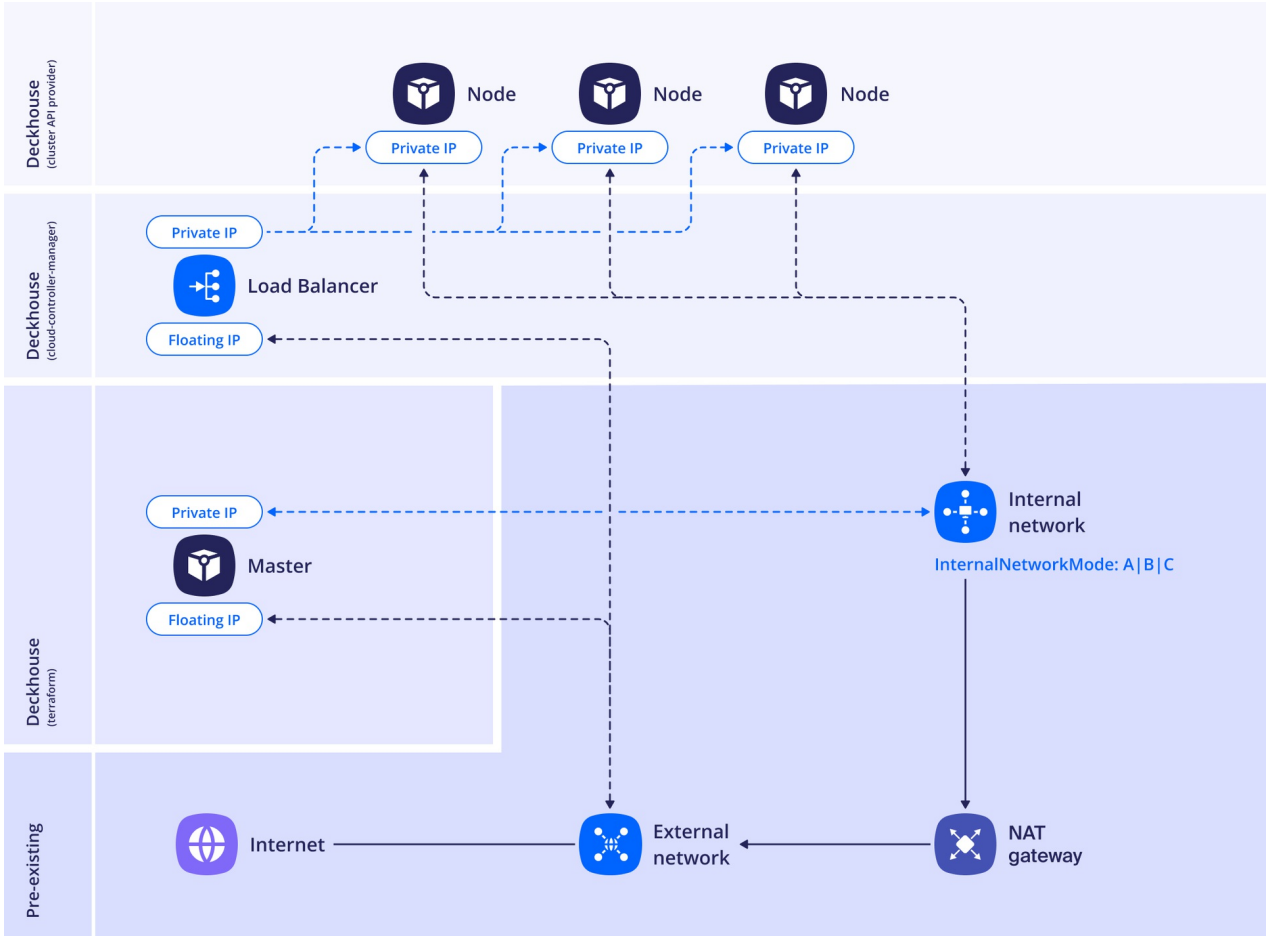
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, true by default.
  # Optional, VXLAN by default, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This layout can be useful if you need to merge a Kubernetes cluster with existing VMs.

> ⚠ This layout does not involve the management of SecurityGroups (it is assumed they were created beforehand). To configure security policies, you must explicitly specify both `additionalSecurityGroups` in the OpenStackClusterConfiguration for the masterNodeGroup and other nodeGroups, and `additionalSecurityGroups` when creating OpenStackInstanceClass in the cluster.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Required, all cluster nodes have to be in the same subnet.
  internalSubnetName: pivot-standard
  # Optional, DirectRoutingWithPortSecurityEnabled by default,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional. If set, it will be used for the load balancer default
  # configuration and ordering the master floating IP.
  externalNetworkName: ext-net
  # Optional, true by default.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Configuration

Integration with OpenStack is performed using the OpenStackClusterConfiguration resource. It defines the configuration of the cloud cluster in OpenStack and is used by the cloud provider when the cluster control plane is hosted in the cloud. The DKP module responsible for the integration is automatically configured based on the selected layout.

To modify the configuration in a running cluster, run the following command:

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

> **i** After changing node-related parameters, run the `dhctl converge` command to apply the changes.

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the OpenStack cloud provider, the instance class is a custom resource called OpenStackInstanceClass, which contains the specific configuration of the VMs.

> **i** An OpenStack cluster is deployed in a single region (`provider.region`). Zones from other regions cannot be used. For details on configuring zones for CloudPermanent and CloudEphemeral nodes, see How do I create NodeGroups in availability zones? section.

> ⚠ When the module settings are changed, **existing Machine objects in the cluster are NOT recreated** (new Machine objects will use the updated parameters). The recreation only occurs if you modify the NodeGroup or the OpenStackInstanceClass parameters.

Configuration examples

The following are two simple configuration examples for the OpenStack cloud provider.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

List of required services

Below is the list of OpenStack services required for DKP to operate in OpenStack:

Service	API version
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* If you need to provision a LoadBalancer.

LoadBalancer configuration

⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

It is recommended to use `loadbalancer.openstack.org/node-selector` to select only the nodes that should be used as targets for the corresponding LoadBalancer.

IngressNginxController example

In this example, the Ingress controller pods are scheduled on frontend nodes, while the `loadbalancer.openstack.org/node-selector` annotation limits the load balancer pool to the same nodes:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Node security policies and configuration

There are many reasons why you might want to restrict or allow incoming or outgoing traffic on cluster VMs. For example:

- Allow connections to cluster nodes from VMs in a different subnet.
- Allow access to specific ports on a static node for application traffic.
- Restrict access to external resources or other VMs in the cloud per security team requirements.

To manage this, you should use additional security groups. Only security groups that were pre-created in the cloud can be used.

Assigning additional security groups to static and master nodes

You can specify additional security groups either during cluster creation or in an existing cluster. In both cases, specify them in the OpenStackClusterConfiguration resource:

- **For master nodes**: Under the `masterNodeGroup` section using the `additionalSecurityGroups` field.
- **For static nodes**: Under the `nodeGroups` section of the corresponding node group configuration and also in the `additionalSecurityGroups` field.

The `additionalSecurityGroups` field is an array of strings representing the names of the security groups.

Assigning additional security groups to ephemeral nodes

To assign additional security groups to ephemeral nodes, specify the `additionalSecurityGroups` parameter in all

relevant OpenStackInstanceClass resources used by these nodes.

Uploading an image to OpenStack

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an openrc file that contains credentials for accessing the OpenStack API.

   > **i** The method of obtaining the openrc file may vary depending on the OpenStack provider. If your provider offers the standard interface for OpenStack, follow this guide to download the file.

3. Alternatively, install the OpenStack CLI tools following the instructions.

   You can also run a container, mounting both the openrc file and the downloaded Ubuntu image:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize environment variables from the openrc file:

   ```
   source /openrc
   ```

5. List the available volume types:

   ```
   openstack volume type list
   ```

   Example output:

   ```
   +--------------------------------------+---------------+-----------+
   | ID                                   | Name          | Is Public |
   +--------------------------------------+---------------+-----------+
   | 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
   | bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
   | 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
   | eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
   | 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
   | a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
   | ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
   | a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
   +--------------------------------------+---------------+-----------+
   ```

6. Create an image and assign the volume type as a property (this is necessary if OpenStack does not support local disks or they are unsuitable for workloads):

   ```
   openstack image create --private --disk-format qcow2 --container-format bare \
     --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
   ```

7. Ensure the image has been created:

   ```
   openstack image show ubuntu-18-04-cloud-amd64
   ```

   Example output:

```
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| Field            | Value
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| checksum         | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-
4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url':
u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap',
u'metadata': {}}]' |
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
```

Verifying security group support by the cloud provider

Run the `openstack security group list` command. If you don’t receive any errors, it means that security groups are supported.

Configuring online volume resize

When resizing a disk via the OpenStack API, Cinder does not pass updated size information to Nova. As a result, the disk size inside the guest OS remains unchanged.

To fix this, you must configure access to the Nova API in the `cinder.conf` file. For example:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

For more details, see the OpenStack-Ansible bug report.

Using rootDiskSize

Disks in OpenStack

A node's disk can be either local or network-attached. In OpenStack terminology:

- A local disk is called an ephemeral disk.
- A network-attached disk is called a persistent disk (Cinder storage).

The ephemeral disk is deleted along with the VM, while the persistent disk remains in the cloud even after the VM is deleted.

- For master nodes, network-attached disks are preferred so the nodes could migrate between hypervisors.
- For ephemeral nodes, local disks are preferred for cost efficiency. However, not all cloud providers support local disks. If not supported, you have to use network-attached disks for ephemeral nodes.

Local disk (ephemeral)	Network-attached disk (persistent)
Deleted with the VM	Remains in cloud and can be reused
Less expensive	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

rootDiskSize parameter

The OpenStackInstanceClass resource includes a `rootDiskSize` parameter. In OpenStack, flavors may also define a disk size.

The resulting disk type depends on the combination of the following parameters:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` not set	*Disk size must be specified*. VM creation will fail without it.	Local disk with size from the flavor.
`rootDiskSize` set	Network-attached disk with size that equals `rootDiskSize`.	Both network-attached (`rootDiskSize`) and local (from flavor). Avoid this setup, as the cloud provider may charge for both.

> **i** In Selectel Cloud, when creating nodes with CloudEphemeral type outside zone A, you must pre-create a flavor with the target disk size. In this case, you do not need to set the `rootDiskSize` parameter.

Recommendations for master nodes and bastion with network-attached disks

- Use a flavor with zero disk size.
- Set `rootDiskSize` in the OpenStackInstanceClass resource.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the type will be taken from `volumeTypeMap`.

Recommendations for ephemeral nodes with local disks

- Use a flavor with a defined disk size.
- Do not use the `rootDiskSize` parameter in OpenStackInstanceClass.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the cloud provider's default disk type will be used.

Checking the disk size in a flavor

Run the following command:

```
openstack flavor show m1.medium-50g -C disk
```

Example output:

```
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

Overriding the cloud provider’s default disk type

If your cloud provider offers multiple volume types, you can specify the default type for an image by setting the `cinder_img_volume_type` property in the image metadata. Any VM created from that image will use the specified volume type.

You can also create a new OpenStack image and upload it.

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Offline disk resizing

If you encounter the following error during volume resizing, reduce the number of replicas in the affected StatefulSet to 0, wait for the volume size to update, and then scale the replicas back to the original value.

```
Warning  VolumeResizeFailed     5s (x11 over 41s)  external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-
9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

Connection and authorization in Amazon Web Services

Requirements

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Before you begin, you must prepare the cloud environment and provide access to AWS resources for Deckhouse Kubernetes Platform (DKP) components that interact with the AWS API.

The `cloud-init` package must be installed on all virtual machines that will be part of the cluster. After booting up, make sure the following services are running:

- `cloud-config.service`
- `cloud-final.service`
- `cloud-init.service`

These services are required for correct EC2 instance initialization and for integration with DKP infrastructure modules such as `cloud-provider-aws` and `machine-controller-manager`.

Accessing the AWS API

To access the AWS API, an IAM user with the required permissions is needed. You must create a JSON policy specification granting access to EC2, ELB, IAM, KMS, and other AWS services.

Example JSON policy with the required permission configuration:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeLaunchConfigurations",
                "autoscaling:DescribeTags",
                "ec2:AllocateAddress",
                "ec2:AssociateAddress",
                "ec2:AssociateRouteTable",
```

```
                "ec2:AttachInternetGateway",
                "ec2:AttachVolume",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CreateInternetGateway",
                "ec2:CreateKeyPair",
                "ec2:CreateNATGateway",
                "ec2:CreateRoute",
                "ec2:CreateRouteTable",
                "ec2:CreateSecurityGroup",
                "ec2:CreateSubnet",
                "ec2:CreateTags",
                "ec2:CreateVolume",
                "ec2:CreateVpc",
                "ec2:DeleteInternetGateway",
                "ec2:DeleteKeyPair",
                "ec2:DeleteNATGateway",
                "ec2:DeleteRoute",
                "ec2:DeleteRouteTable",
                "ec2:DeleteSecurityGroup",
                "ec2:DeleteSubnet",
                "ec2:DeleteTags",
                "ec2:DeleteVolume",
                "ec2:DeleteVpc",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAddresses",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeImages",
                "ec2:DescribeInstanceAttribute",
                "ec2:DescribeInstanceCreditSpecifications",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceTypes",
                "ec2:DescribeInstanceTopology",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeNatGateways",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeRegions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSecurityGroupRules",
                "ec2:DescribeSubnets",
                "ec2:DescribeTags",
                "ec2:DescribeVolumesModifications",
                "ec2:DescribeVolumes",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcClassicLink",
                "ec2:DescribeVpcClassicLinkDnsSupport",
                "ec2:DescribeVpcs",
                "ec2:DetachInternetGateway",
                "ec2:DetachVolume",
                "ec2:DisassociateAddress",
                "ec2:DisassociateRouteTable",
                "ec2:ImportKeyPair",
                "ec2:ModifyInstanceAttribute",
                "ec2:ModifySubnetAttribute",
                "ec2:ModifyVolume",
                "ec2:ModifyVpcAttribute",
                "ec2:ReleaseAddress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress",
                "ec2:RunInstances",
                "ec2:TerminateInstances",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:CreateVpcPeeringConnection",
                "ec2:DeleteVpcPeeringConnection",
                "ec2:AcceptVpcPeeringConnection",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "elasticloadbalancing:AddTags",
```

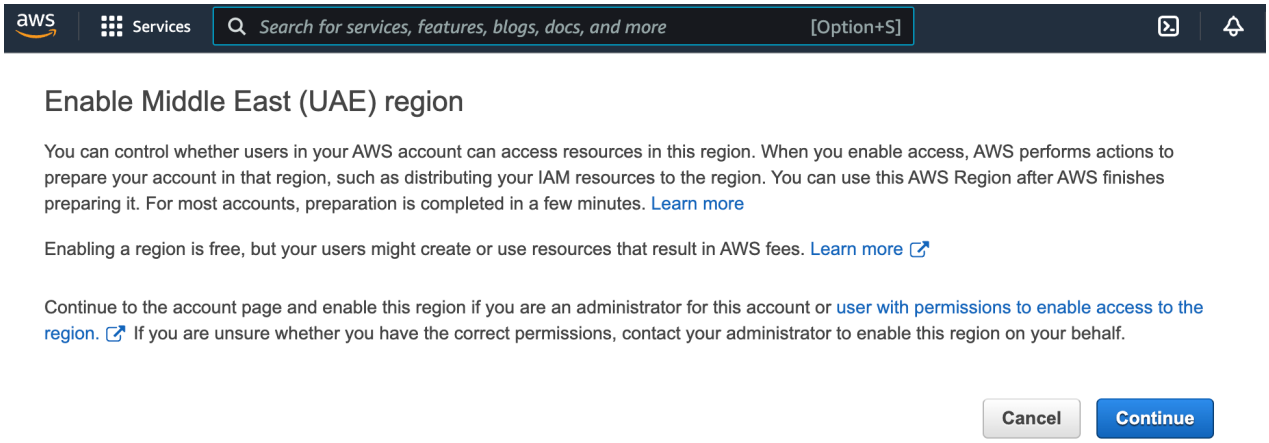
```
                "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
                "elasticloadbalancing:AttachLoadBalancerToSubnets",
                "elasticloadbalancing:ConfigureHealthCheck",
                "elasticloadbalancing:CreateListener",
                "elasticloadbalancing:CreateLoadBalancer",
                "elasticloadbalancing:CreateLoadBalancerListeners",
                "elasticloadbalancing:CreateLoadBalancerPolicy",
                "elasticloadbalancing:CreateTargetGroup",
                "elasticloadbalancing:DeleteListener",
                "elasticloadbalancing:DeleteLoadBalancer",
                "elasticloadbalancing:DeleteLoadBalancerListeners",
                "elasticloadbalancing:DeleteTargetGroup",
                "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
                "elasticloadbalancing:DeregisterTargets",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeLoadBalancerAttributes",
                "elasticloadbalancing:DescribeLoadBalancerPolicies",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeTargetHealth",
                "elasticloadbalancing:DetachLoadBalancerFromSubnets",
                "elasticloadbalancing:ModifyListener",
                "elasticloadbalancing:ModifyLoadBalancerAttributes",
                "elasticloadbalancing:ModifyTargetGroup",
                "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
                "elasticloadbalancing:RegisterTargets",
                "elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer",
                "elasticloadbalancing:SetLoadBalancerPoliciesOfListener",
                "iam:AddRoleToInstanceProfile",
                "iam:CreateInstanceProfile",
                "iam:CreateRole",
                "iam:CreateServiceLinkedRole",
                "iam:DeleteInstanceProfile",
                "iam:DeleteRole",
                "iam:DeleteRolePolicy",
                "iam:GetInstanceProfile",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListInstanceProfilesForRole",
                "iam:ListRolePolicies",
                "iam:ListAttachedRolePolicies",
                "iam:PassRole",
                "iam:PutRolePolicy",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:TagRole",
                "kms:DescribeKey",
                "sts:GetCallerIdentity"
            ],
            "Resource": "*"
        }
    ]
}
```

Configuring IAM via the AWS web console

1. Create policies:
   - Go to **Identity and Access Management (IAM)** → **Policies**.
   - Click **Create policy**.
   - Switch to the **JSON** tab and paste the prepared JSON specification.
   - Click **Next: Tags**, then **Next: Review**.
   - Provide a name for the policy, for example, D8CloudProviderAWS.
   - Click **Create policy**.
2. Create a user:
   - Go to **IAM** → **Users**.
   - Click **Add users**.
   - Enter a username, for example, deckhouse.
   - Under **Select AWS credential type**, choose **Access key** → **Programmatic access**.
   - Click **Next: Permissions**.
3. Attach policies to the user:
   - Choose **Attach existing policies directly**.
   - Enter the name of the policy created earlier in the filter field and check the box next to it.
   - Click **Next: Tags**, then **Next: Review**.
   - Click **Create user**.

> **i** Save the generated `Access key ID` and `Secret access key`. You will need these to configure DKP.

Ensure that your account (and the created user) has access to the required regions. To verify this, select the region from the dropdown menu in the upper-right corner. If the region switches successfully, access is available. If not, you may see a message like this:



Click **Continue** to enable the region.

Configuring IAM via AWS CLI

You can also configure IAM using the command-line interface. Follow these steps:

1. Save the JSON specification to a file named `policy.json`:

   ```
   cat > policy.json << EOF
   <Policy JSON spec>
   EOF
   ```

2. Create a new policy named `D8CloudProviderAWS` with the `ARN` details using the specification from the `policy.json`:

   ```
   aws iam create-policy --policy-name D8CloudProviderAWS --policy-document file://policy.json
   ```

   Example output:

   ```
   {
       "Policy": {
           "PolicyName": "D8CloudProviderAWS",
           "PolicyId": "AAA",
           "Arn": "arn:aws:iam::123:policy/D8CloudProviderAWS",
           "Path": "/",
           "DefaultVersionId": "v1",
           "AttachmentCount": 0,
           "PermissionsBoundaryUsageCount": 0,
           "IsAttachable": true,
           "CreateDate": "2020-08-27T02:52:06+00:00",
           "UpdateDate": "2020-08-27T02:52:06+00:00"
       }
   }
   ```

3. Create a new user:

   ```
   aws iam create-user --user-name deckhouse
   ```

   Example output:

```
{
    "User": {
        "Path": "/",
        "UserName": "deckhouse",
        "UserId": "AAAXXX",
        "Arn": "arn:aws:iam::123:user/deckhouse",
        "CreateDate": "2020-08-27T03:05:42+00:00"
    }
}
```

4. Grant access to the API and save `AccessKeyId` and `SecretAccessKey`:

   ```
   aws iam create-access-key --user-name deckhouse
   ```

   Example output:

   ```
   {
       "AccessKey": {
           "UserName": "deckhouse",
           "AccessKeyId": "XXXYYY",
           "Status": "Active",
           "SecretAccessKey": "ZZZzzz",
           "CreateDate": "2020-08-27T03:06:22+00:00"
       }
   }
   ```

5. Attach the policy to the user combining `User` and `Policy`:

   ```
   aws iam attach-user-policy --user-name username --policy-arn arn:aws:iam::123:policy/D8CloudProviderAWS
   ```

Configuring IAM via Terraform

Example Terraform resource configuration:

```
resource "aws_iam_user" "user" {
  name = "deckhouse"
}

resource "aws_iam_access_key" "user" {
  user = aws_iam_user.user.name
}

resource "aws_iam_policy" "policy" {
  name       = "D8CloudProviderAWS"
  path       = "/"
  description = "Deckhouse policy"

  policy = <<EOF
<JSON policy specification>
EOF
}

resource "aws_iam_user_policy_attachment" "policy-attachment" {
  user      = aws_iam_user.user.name
  policy_arn = aws_iam_policy.policy.arn
}
```

Using a custom IAM role for nodes (iamNodeRole)

The `iamNodeRole` parameter in the AWSClusterConfiguration resource lets you override the default IAM role that DKP assigns to all EC2 instances of the cluster nodes.

By default, DKP creates and assigns an IAM role named `<prefix>-node`, where `<prefix>` is the value of the global `cloud.prefix` parameter. This role includes the minimum permissions required for node operation.

You can specify the IAM role name in the `iamNodeRole` parameter if you want to:

- Use a pre-created IAM role managed manually.
- Grant additional permissions to nodes (for example, access to specific AWS services).

> **i** The specified role must include all permissions required by DKP. Without them, the nodes will not function properly. In particular, it must include the same permissions as the default `<prefix>-node` role, even if they are assigned manually.

For more information on IAM roles for AWS EC2, refer to the AWS documentation.

Additional resources

- Module `cloud-provider-aws` documentation

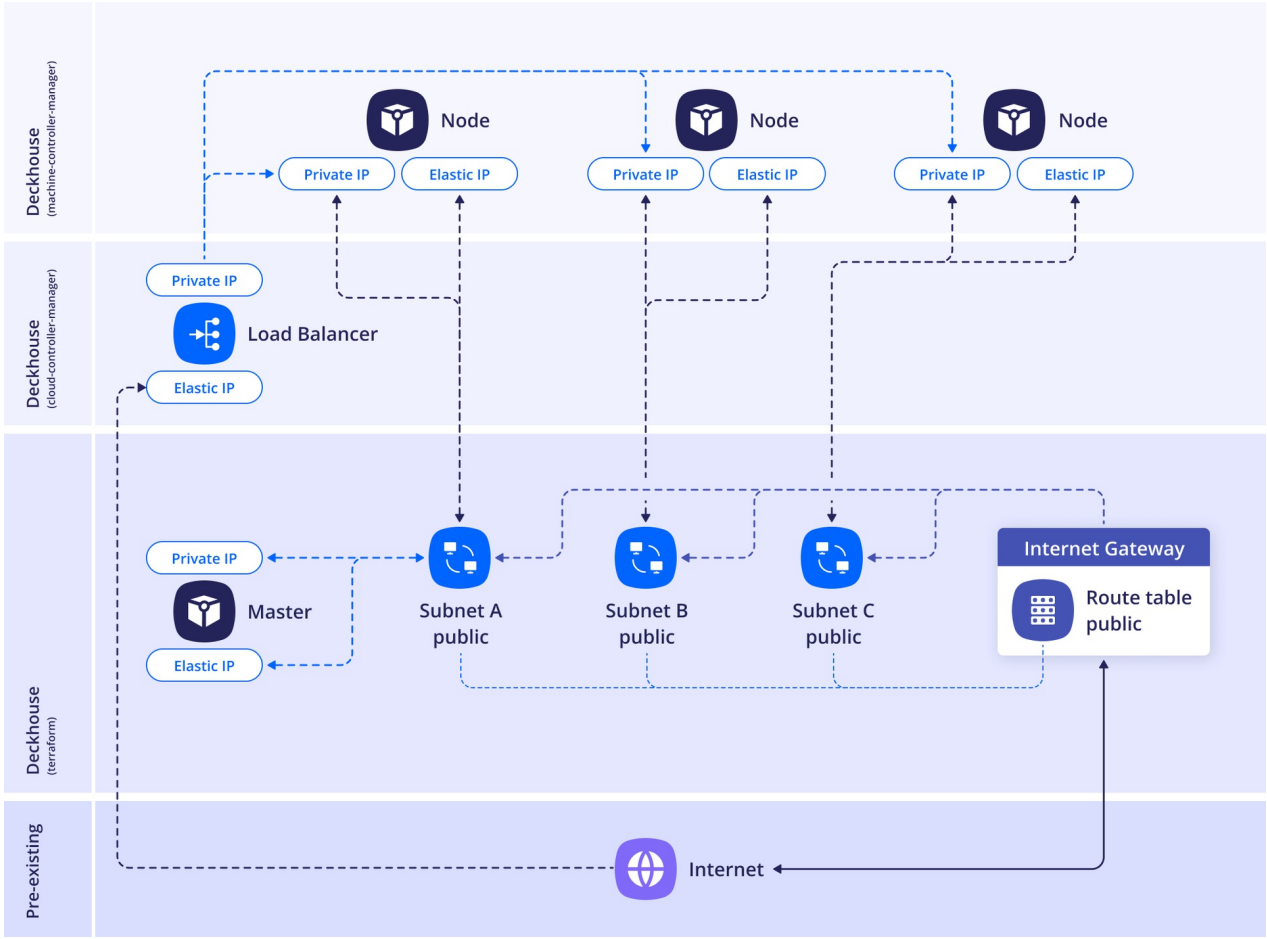
Layouts and configuration in Amazon Web Services

This section describes cluster layouts in AWS infrastructure and their related parameters. The selected layout affects how public IP addresses are assigned, NAT usage, and how nodes are accessed.

WithoutNAT

This is the recommended layout.

Each node is assigned a public IP address (Elastic IP), and no NAT Gateway is used. This layout allows direct access to nodes via public IP addresses and simplifies outgoing traffic routing.



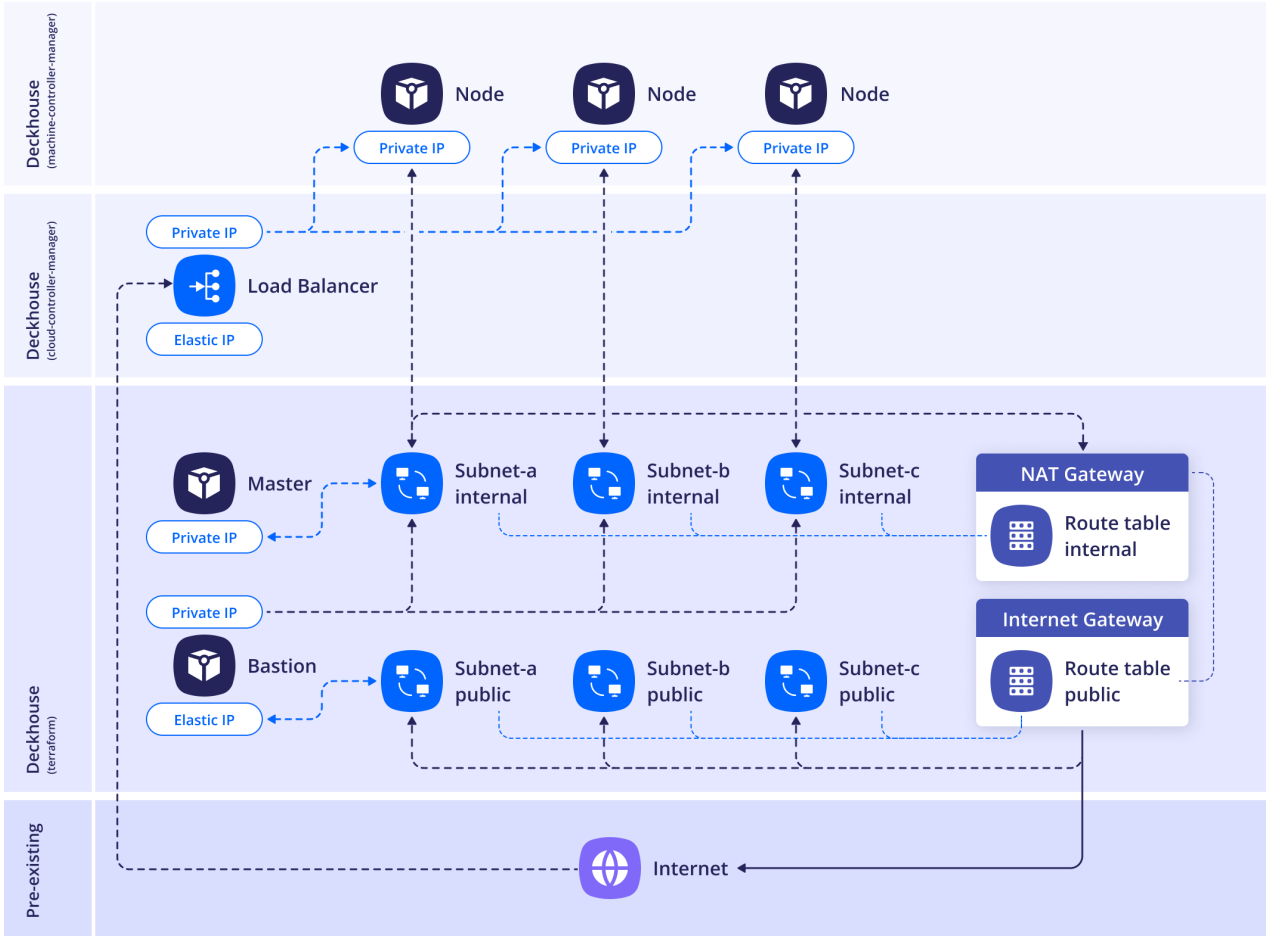
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithoutNAT
vpcNetworkCIDR: "10.241.0.0/16"
nodeNetworkCIDR: "10.241.32.0/20"
sshPublicKey: <SSH_PUBLIC_KEY>
provider:
  providerAccessKeyId: '<AWS_ACCESS_KEY>'
  providerSecretAccessKey: '<AWS_SECRET_ACCESS_KEY>'
  region: eu-central-1
masterNodeGroup:
  # Number of master nodes.
  # If you specify more than one master node, an etcd cluster will be created automatically.
  replicas: 1
  instanceClass:
    # Instance type to use.
    instanceType: m5.xlarge
    # ID of the Amazon Machine Image (AMI).
    # AMI catalog can be found in the AWS Console: EC2 → AMI Catalog.
    ami: ami-0aad10862ade98f27
    # Disk size for the master node VM.
    diskSizeGb: 30
    # Disk type to use for the master node VM.
    diskType: gp3
nodeGroups:
  - name: mydb
    nodeTemplate:
      labels:
        node-role.kubernetes.io/mydb: ""
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aad10862ade98f27
    additionalTags:
      backup: srv1
tags:
  team: torpedo
```

WithNAT

> ⚠ In this layout, the NAT Gateway is always created in zone `a`. If cluster nodes are placed in other zones, a failure in zone a may cause the cluster to become unavailable. A bastion host is required to connect to the nodes.

In this layout, a NAT Gateway is used for accessing the internet, and nodes do not receive public IP addresses. Access to nodes is only possible through a bastion host, which is placed in a separate subnet.



Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithNAT
provider:
  providerAccessKeyId: '<AWS_ACCESS_KEY>'
  providerSecretAccessKey: '<AWS_SECRET_ACCESS_KEY>'
  region: eu-central-1
withNAT:
  bastionInstance:
    zone: eu-central-1a
    instanceClass:
      instanceType: m5.large
      ami: ami-0aad10862ade98f27
      diskType: gp3
masterNodeGroup:
  # Number of master nodes.
  # If you specify more than one master node, an etcd cluster will be created automatically.
  replicas: 1
  instanceClass:
    # Instance type to use.
    instanceType: m5.xlarge
    # ID of the Amazon Machine Image (AMI).
    # AMI catalog can be found in the AWS Console: EC2 → AMI Catalog.
    ami: ami-0aad10862ade98f27
    # Disk size for the master node VM.
    diskSizeGb: 30
    # Disk type to use for the master node VM.
    diskType: gp3
nodeGroups:
  - name: mydb
    nodeTemplate:
      labels:
        node-role.kubernetes.io/mydb: ""
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aad10862ade98f27
    additionalTags:
      backup: me
vpcNetworkCIDR: "10.241.0.0/16"
nodeNetworkCIDR: "10.241.32.0/20"
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  team: torpedo
```

Defining AWSClusterConfiguration

The AWSClusterConfiguration resource describes the cluster settings and is used by Deckhouse Kubernetes Platform (DKP) to:

- Define the layout and network CIDRs.
- Configure master and worker nodes.
- Specify AWS API connection parameters (such as access keys and region).
- Define common and custom tags.
- Configure the bastion host (for the WithNAT layout).

Required fields:

- `apiVersion`: Must be set to `deckhouse.io/v1`.
- `kind`: Always AWSClusterConfiguration.

Example resource header:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
```

To edit this resource in a running cluster, run the following command:

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

After editing, apply the changes with the following command:

```
dhctl converge
```

Internal addressing and subnets

The `nodeNetworkCIDR` parameter defines the address range to be split among availability zones. This range must match or be a subset of `vpcNetworkCIDR`. Subnets are automatically created based on the number of zones in the region.

Example:

```
nodeNetworkCIDR: 10.241.1.0/20
vpcNetworkCIDR: 10.241.0.0/16
```

Security groups

AWS security groups are used to manage incoming and outgoing traffic to virtual machines. In DKP, they can be used to:

- Allow access to cluster nodes from other subnets.
- Open access to applications running on static nodes.
- Restrict or allow access to external resources based on security policies.

> **i** DKP does not create security groups automatically. In the cluster configuration, you must specify existing security groups created manually via the AWS Console or other means.

You can assign additional security groups in the following cases:

Node type	Where to configure
Master nodes	In the `masterNodeGroup.instanceClass.additionalSecurityGroups` field of the AWSClusterConfiguration resource
Static worker nodes	In the `nodeGroups[].instanceClass.additionalSecurityGroups` field of the same resource
Ephemeral nodes	In the `spec.additionalSecurityGroups` field of the AWSInstanceClass object

In all cases, the `additionalSecurityGroups` parameter must be an array of strings (AWS security group IDs or names).

If `disableDefaultSecurityGroup: true` is set, the default security groups will not be created.

When using `disableDefaultSecurityGroup: true`, you must manually create all required security groups to allow access to cluster nodes. Additionally, you must explicitly specify them in the following parameters:

- `additionalSecurityGroups` in the `masterNodeGroup` section of the AWSClusterConfiguration resource;
- `additionalSecurityGroups` in the AWSInstanceClass resource;
- `additionalSecurityGroups` in the `nodeGroups.instanceClass` section.

To configure the security groups used by load balancers, specify them using the `service.beta.kubernetes.io/aws-load-balancer-security-groups` annotation.

Setting up VPC peering

The following example shows how to set up VPC peering between two sample VPCs: `vpc-a` and `vpc-b`.

> **⚠** The IPv4 CIDR blocks of both VPCs must be different.

To configure peering, follow these steps:

1. Go to the region where `vpc-a` is working.
2. Navigate to VPC → VPC Peering Connections → Create Peering Connection, then configure the peering:
   - Name: `vpc-a-vpc-b`.
   - Fill in “Local” and “Another VPC” sections.
3. Go to the region where `vpc-b` is working.
4. Navigate to VPC → VPC Peering Connections.
5. Select the created connection and click **Accept Request**.
6. In `vpc-a`, add routes to the `vpc-b` CIDR in all route tables via the peering connection.
7. In `vpc-b`, add routes to the `vpc-a` CIDR in all route tables via the peering connection.

Configuring access via bastion host

To access nodes in private subnets, use the `withNAT.bastionInstance` parameter in AWSClusterConfiguration. The bastion host is provisioned with the infrastructure according to the `instanceClass` settings.

The following scenarios are supported:

- The bastion host already exists in an external VPC:
  1. Run the base infrastructure bootstrap: `dhctl bootstrap-phase base-infra`.
  2. Set up VPC peering between the external and new VPC.
  3. Continue installation using the bastion host: `dhctl bootstrap --ssh-bastion...`.
- The bastion host is required in the new VPC:
  1. Run the base infrastructure bootstrap: `dhctl bootstrap-phase base-infra`.

2. Manually launch the bastion host in the `<prefix>-public-0` subnet.
3. Continue installation using the bastion host: `dhctl bootstrap --ssh-bastion...`.

Creating a cluster in a new VPC with an existing bastion host

1. Bootstrap the cluster's base infrastructure:

   ```
   dhctl bootstrap-phase base-infra --config config
   ```

2. Set up VPC peering as described above.
3. Continue cluster installation. When prompted about the Terraform cache, answer with `y`:

   ```
   dhctl bootstrap --config config --ssh-...
   ```

Creating a cluster in a new VPC and deploying a bastion host

1. Bootstrap the cluster's base infrastructure:

   ```
   dhctl bootstrap-phase base-infra --config config
   ```

2. Manually launch a bastion host in the `<prefix>-public-0` subnet.
3. Continue cluster installation. When prompted about the Terraform cache, answer with `y`:

   ```
   dhctl bootstrap --config config --ssh-...
   ```

Using an existing VPC (existingVPCID)

The `existingVPCID` parameter in the AWSClusterConfiguration resource lets you use an existing VPC for DKP cluster deployment instead of automatically creating a new one.

This may be useful when:

- Your AWS infrastructure is already partially deployed.
- You need to integrate with other services or resources in the same VPC.
- Security policies or architectural guidelines prevent automatic VPC creation.

> ⚠ If the existing VPC already has an Internet Gateway, the base infrastructure bootstrap will fail. Reusing an existing Internet Gateway is not supported in the current DKP version.

Compatibility with other parameters:

- If `existingVPCID` is specified, do not specify `vpcNetworkCIDR` as they are mutually exclusive.
- The `nodeNetworkCIDR` parameter can (and should) be specified. It must be a subset of the existing VPC.

Additional resources

- Module `cloud-provider-aws` provider configuration
- Module `cloud-provider-aws` custom resources

Storage and load balancing in Amazon Web Services

This section covers the following additional aspects of integrating Deckhouse Kubernetes Platform (DKP) with AWS:

- Connecting cloud disks using CSI.
- Automatic StorageClass creation.
- Using LoadBalancer.
- Accessing via bastion host.
- Connecting manually created CloudStatic nodes.

Storage (CSI and StorageClass)

DKP integrates with AWS storage via CSI. This allows the cluster to automatically provision and attach disks to its nodes.

StorageClasses are created automatically for the following disk types:

- `gp3`
- `gp2`
- `sc1`
- `st1`

Disk types `io1` and `io2` are also supported with IOPS and throughput settings in ModuleConfig.

To exclude unnecessary StorageClasses from the cluster, specify filters in `settings.storageClass.exclude`:

```
settings:
  storageClass:
    exclude:
    - sc.*
    - st1
```

You can also define and configure custom StorageClasses explicitly, including parameters such as `iops`, `throughput`, and `type`:

```
settings:
  storageClass:
    provision:
    - name: fast-io
      type: io2
      iopsPerGB: "50"
```

Expanding volume size

To resize a volume (for example, when running low on disk space), follow these steps:

1. Modify the `spec.resources.requests.storage` parameter in the corresponding PersistentVolumeClaim object. The operation is performed automatically and usually takes less than a minute. You can monitor progress with the following command:

   ```
   kubectl describe pvc <claim-name>
   ```

2. After resizing, wait at least 6 hours and ensure the volume status is `in-use` or `available`. Only then you can safely perform additional size changes. See AWS documentation for details.

Load balancing

DKP supports LoadBalancer Services via AWS Load Balancer Controller.

To control how AWS resources are created, use annotations on the Service object:

```
metadata:
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
    service.beta.kubernetes.io/aws-load-balancer-subnets: "subnet-12345,subnet-67890"
```

Where:

- `service.beta.kubernetes.io/aws-load-balancer-type`: If set to none, only a target group is created (no LoadBalancer).
- `service.beta.kubernetes.io/aws-load-balancer-backend-protocol`: Only applicable when `aws-load-balancer-type` is `none`. Specifies the protocol for communication with the target group. Supported values:
  - `tcp` (default)
  - `tls`
  - `http`
  - `https`

> **i**
>
> When you change this annotation, `cloud-controller-manager` will attempt to recreate the target group. If it's already associated with an NLB or ALB, the deletion will fail, and the controller will enter a retry loop. To avoid this, manually detach the load balancer from the group.
>
> If Ingress nodes are not available in all zones, explicitly specify subnets using `aws-load-balancer-subnets`.

Configuring LoadBalancer when Ingress nodes are not in all zones

If your AWS cluster does not have Ingress nodes in every zone, you must explicitly list the subnets to be used by the Load Balancer via the `service.beta.kubernetes.io/aws-load-balancer-subnets` annotation:

```
metadata:
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-subnets: "subnet-foo,subnet-bar"
```

This is especially important when manually configuring Ingress controllers or using non-standard node layout.

To retrieve the current list of subnets used in your DKP setup, run the following:

```
kubectl -n d8-system exec svc/deckhouse-leader -c deckhouse \
  -- deckhouse-controller module values cloud-provider-aws -o json | \
  jq -r '.cloudProviderAws.internal.zoneToSubnetIdMap'
```

Connecting CloudStatic nodes

To connect manually created EC2 instances to a DKP cluster, follow these steps:

1. Attach the IAM role `<prefix>-node`.
2. Attach the security group `<prefix>-node`.
3. Add the following tags:

```
"kubernetes.io/cluster/<cluster_uuid>" = "shared"
"kubernetes.io/cluster/<prefix>" = "shared"
```

To get the `cluster_uuid`, use the following command:

```
kubectl -n kube-system get cm d8-cluster-uuid -o json | jq -r '.data."cluster-uuid"'
```

To get the `prefix`, use the following command:

```
kubectl -n kube-system get secret d8-cluster-configuration -o json | \
  jq -r '.data."cluster-configuration.yaml"' | base64 -d | grep prefix
```

Additional resources

- Module `cloud-provider-aws` parameters

Integration with Amazon Web Services

Working with Amazon Elastic Container Registry (ECR)

To allow cluster nodes to access private Amazon ECR repositories, follow these steps:

1. Define read permissions for container images in the repository policies. It's important that the `Principal` field includes the IAM Role attached to the Deckhouse Kubernetes Platform (DKP) nodes.

   Example policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RepositoryRead",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::xxx:role/xxx-node"
      },
      "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:DescribeRepositories",
        "ecr:GetAuthorizationToken",
        "ecr:GetDownloadUrlForLayer",
        "ecr:ListImages",
        "ecr:ListTagsForResource"
      ]
    }
  ]
}
```

2. Set this policy via the AWS Console. Go to **Amazon ECR** → **Private registry** → **Repositories** → select the desired repository → **Permissions**.
3. Add `ecr:GetAuthorizationToken` to `additionalRolePolicies` in your AWSClusterConfiguration so that nodes can automatically retrieve an authorization token to pull images:

```
additionalRolePolicies:
  - ecr:GetAuthorizationToken
  - ecr:BatchGetImage
  - ecr:DescribeRepositories
```

The additionalRolePolicies parameter lets you extend the set of IAM actions granted to EC2 instances managed by DKP. This is particularly useful if access to the following is required:

- Amazon ECR
- Amazon S3
- Other AWS services that require specific permissions

If this parameter is not set, IAM roles will only include the basic actions:

```
ec2:DescribeTags
ec2:DescribeInstances
```

Additional resources

- Module `cloud-provider-aws` provider configuration

Connection and authorization in Microsoft Azure

Requirements

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

To ensure Deckhouse Kubernetes Platform (DKP) works correctly with Microsoft Azure, the following conditions must be met:

- The selected region must support Availability Zones.
- All virtual machines must have the `cloud-init` package installed. After the VMs start, the following services must be active:
  - `cloud-config.service`
  - `cloud-final.service`
  - `cloud-init.service`

Accessing the Azure API

To manage Azure resources from DKP, you need a service account with the `Contributor` role within the appropriate subscription. Follow these steps:

1. Install Azure CLI and log in:

   ```
   export SUBSCRIPTION_ID=$(az login | jq -r '.[0].id')
   ```

2. Create a service account:

   ```
   az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/$SUBSCRIPTION_ID" --name
   "DeckhouseCANDI"
   ```

   Example output:

   ```
   {
    "appId": "<CLIENT_ID>",
    "displayName": "DeckhouseCANDI",
    "password": "<CLIENT_SECRET>",
    "tenant": "<TENANT_ID>"
   }
   ```

   Specify the output values in the AzureClusterConfiguration object:

Field	Value from the output
`clientId`	`appId`
`clientSecret`	`password`
`tenantId`	`tenant`

3. Authenticate in Azure CLI using the created service account's credentials:

   ```
   az login --service-principal -u <CLIENT_ID> -p <CLIENT_SECRET> --tenant <TENANT_ID>
   ```

ℹ By default, the `clientSecret` is valid for 1 year, and no automatic renewal is supported. To set a longer expiration period, refer to the official Azure documentation.

Additional resources

- Module `cloud-provider-azure` provider configuration

Layouts and configuration in Microsoft Azure

This section describes the cluster deployment layout in Microsoft Azure infrastructure and the associated parameters.

Standard

Standard is the supported deployment layout with the following characteristics:

- A separate resource group is created for the cluster.
- Each instance is assigned a public IP address by default (used only for accessing the internet).
- Up to 64,000 SNAT ports are available per public IP address.
- A NAT Gateway is supported and billable. It allows using static public IP addresses for SNAT.
- Public IPs can be assigned to both master nodes and nodes created via Terraform.
- If the master node does not have a public IP, a bastion host and VNet peering between clusters are required.
- Peering between the cluster’s VNet and other VNets is supported.

Example configuration:

```
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"      # Required.
vNetCIDR: 10.50.0.0/16               # Required.
subnetCIDR: 10.50.0.0/24             # Required.
standard:
  natGatewayPublicIpCount: 1         # Optional (0 by default).
masterNodeGroup:
  replicas: 1
  zones: ["1"]                      # Optional (["1"] by default).
  instanceClass:
    machineSize: Standard_F4         # Required.
    diskSizeGb: 32
    urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140  # Required.
    enableExternalIP: false          # Optional (true by default).
provider:
  subscriptionId: "<SUBSCRIPTION_ID>"  # Required.
  clientId: "<CLIENT_ID>"            # Required.
  clientSecret: "<CLIENT_SECRET>"     # Required.
  tenantId: "<TENANT_ID>"            # Required.
  location: "westeurope"             # Required.
peeredVNets:                        # Optional.
  - resourceGroupName: kube-bastion   # Required.
    vnetName: kube-bastion-vnet      # Required.
```

Storage and load balancing in Microsoft Azure

Storage

When running on Azure, Deckhouse Kubernetes Platform (DKP) automatically creates the following StorageClass resources:

Name	Disk type
`managed-standard`	Standard_LRS
`managed-standard-ssd`	StandardSSD_LRS
`managed-premium`	Premium_LRS

You can additionally:

- Disable unneeded storage classes using the `exclude` parameter.
- Define custom StorageClass resources with the target throughput and IOPS settings.

Example configuration in a ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-azure
spec:
  version: 1
  enabled: true
  settings:
    storageClass:
      provision:
      - name: managed-ultra-ssd
        type: UltraSSD_LRS
        diskIOPSReadWrite: 600
        diskMBpsReadWrite: 150
      exclude:
      - managed-standard.*
      - managed-premium
```

Load balancing

DKP automatically creates LoadBalancer resources in Azure when using Kubernetes Service objects of the LoadBalancer type.

Additional features:

- NAT Gateway is supported. You can explicitly define the number of public IP addresses for SNAT using the `natGatewayPublicIpCount` parameter.
- VNet peering can be configured, including with a bastion host or other VNets in the cloud.
- Service Endpoints are supported. They provide secure and direct connections to Azure services without using public IPs. For more details, refer to Integration with Microsoft Azure services.

Additional resources

- Module `cloud-provider-azure` parameters
- Module `cloud-provider-azure` provider configuration

Integration with Microsoft Azure services

Deckhouse Kubernetes Platform (DKP) leverages Azure cloud features for full integration with Kubernetes. When running a cluster on Azure, it automatically:

- Creates network routes for the PodNetwork.
- Configures external LoadBalancers for Kubernetes Service objects.
- Removes nodes from the cluster that no longer exist in the cloud.
- Updates node metadata to reflect the current configuration.
- Provisions disks for nodes via CSI.
- Connects the required CNI network (a simple bridge is used).
- Makes AzureInstanceClass definitions available for virtual machines, which can be used later in NodeGroup configurations.

i All outgoing traffic from the cluster comes through LoadBalancers. If no LoadBalancer is configured to handle UDP traffic, all outgoing UDP traffic will be blocked, which can affect NTP utilities such as `ntpdate`, `chrony`, etc. To resolve this, manually add a rule for any UDP port to an existing LoadBalancer or create a Kubernetes Service of the LoadBalancer type with a UDP port.

Support for Service Endpoints

DKP supports connections to Azure services via Service Endpoints. These endpoints:

- Allow access to Azure services without using public IP addresses.
- Route traffic over optimized Azure backbone infrastructure.
- Simplify access control and improve security.

List of supported Service Endpoints:

```
Microsoft.AzureActiveDirectory
Microsoft.AzureCosmosDB
Microsoft.ContainerRegistry
Microsoft.CognitiveServices
Microsoft.EventHub
Microsoft.KeyVault
Microsoft.ServiceBus
Microsoft.Sql
Microsoft.Storage
Microsoft.Storage.Global
Microsoft.Web
```

Specify the required services using the `serviceEndpoints` parameter in the AzureClusterConfiguration object.

Additional resources

- Module `cloud-provider-azure` custom resources
- Module `cloud-provider-azure` provider configuration
- Module `node-manager` custom resources

Connection and authorization in Google Cloud Platform

To manage Google Cloud resources using Deckhouse Kubernetes Platform, you need to create a Service Account.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Creating a service account

You can find detailed instructions on how to create a service account in the official Google Cloud documentation.

⚠ A created `service account key` cannot be recovered. If lost, it must be deleted and recreated.

Setup via Google Cloud console

Go to the Google Cloud console, select your project, and create a new service account (you can also choose an existing one).

The created service account must be assigned the following roles:

```
Compute Admin
Service Account User
Network Management Admin
```

You can assign roles during service account creation or modify them later.

To generate the `service account key` in JSON format, on the service accounts page, click the three vertical dots in the **Actions** column and select **Manage keys**. Then click **Add key** → **Create new key** → **Key type** → **JSON**.

Setup via Google Cloud CLI

Install and initialize the Google Cloud CLI by following the official instructions.

To create a service account using the CLI, follow these steps:

1. Export environment variables:

   ```
   export PROJECT_ID=sandbox
   export SERVICE_ACCOUNT_NAME=deckhouse
   ```

2. Set the project:

   ```
   gcloud config set project $PROJECT_ID
   ```

3. Create a service account:

   ```
   gcloud iam service-accounts create $SERVICE_ACCOUNT_NAME
   ```

4. Assign roles to the created service account:

   ```
   for role in roles/compute.admin roles/iam.serviceAccountUser roles/networkmanagement.admin;
   do gcloud projects add-iam-policy-binding ${PROJECT_ID} --
   member=serviceAccount:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
      --role=${role}; done
   ```

   A list of required roles:

   ```
   roles/compute.admin
   roles/iam.serviceAccountUser
   roles/networkmanagement.admin
   ```

5. Verify the assigned roles:

```
gcloud projects get-iam-policy ${PROJECT_ID} --flatten="bindings[].members" --
format='table(bindings.role)' \
     --filter="bindings.members:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com"
```

6. Generate a `service account key`:

```
gcloud iam service-accounts keys create --iam-account
${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
     ~/service-account-key-${PROJECT_ID}-${SERVICE_ACCOUNT_NAME}.json
```

Using the service account

The generated `service account key` must be specified in the `provider.serviceAccountJSON: "<SERVICE_ACCOUNT_JSON>"` section of the GCPClusterConfiguration resource.

Example:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
sshKey: "<SSH_PUBLIC_KEY>"
subnetworkCIDR: 10.36.0.0/24
masterNodeGroup:
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20240523a
    diskSizeGb: 50
nodeGroups:
- name: static
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20240523a
    diskSizeGb: 50
    additionalNetworkTags:
    - tag1
    additionalLabels:
      kube-node: static
provider:
  region: europe-west3
  serviceAccountJSON: "<SERVICE_ACCOUNT_JSON>"
```

Additional resources

- Module `cloud-provider-gcp` provider configuration

Layouts and configuration in Google Cloud Platform

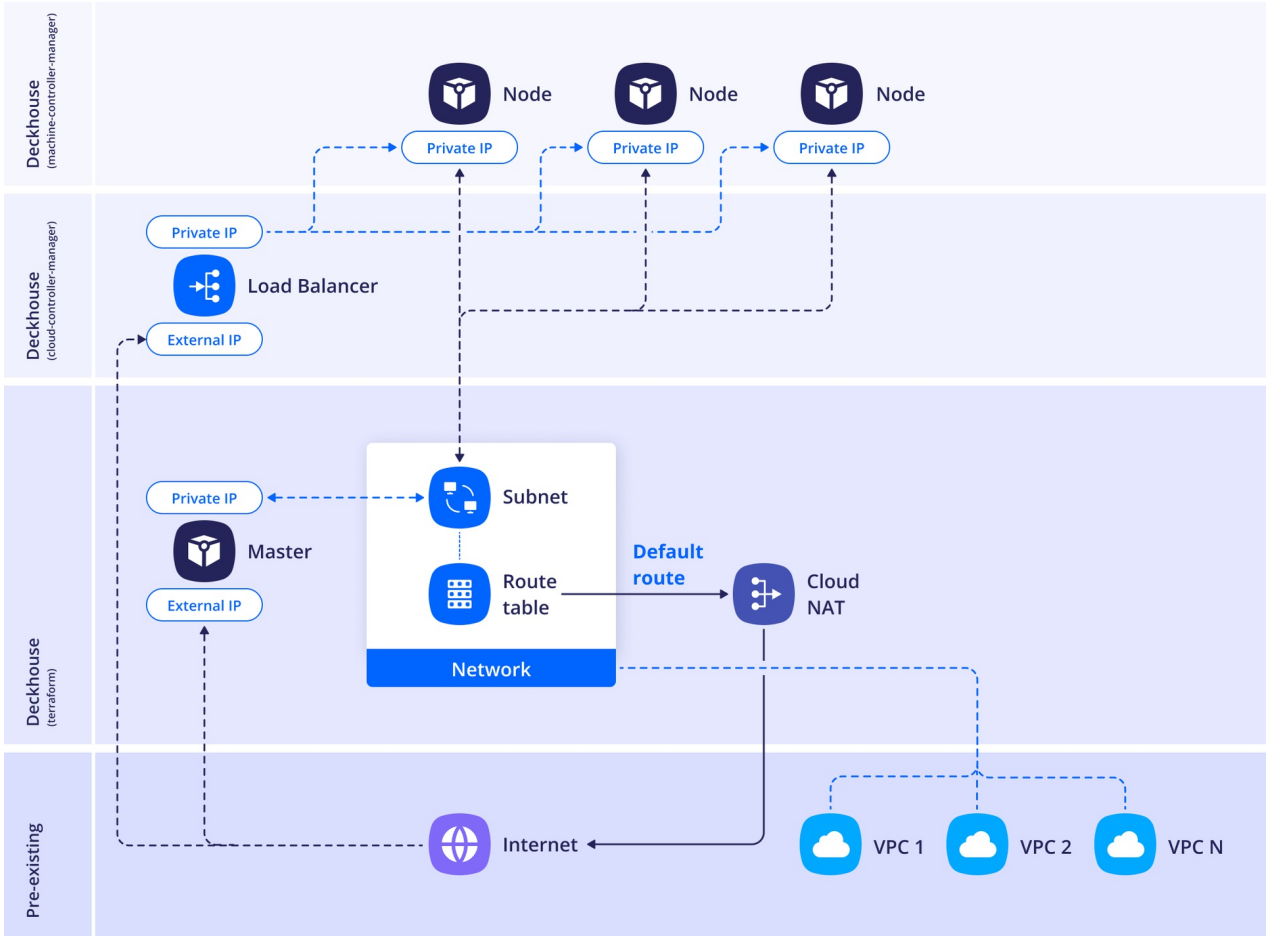
Layouts

This section describes the possible node placement layouts in Google Cloud Platform (GCP) infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Deckhouse Kubernetes Platform (DKP) supports two layouts for deploying resources in GCP.

Standard

- A dedicated VPC is created for the cluster along with Cloud NAT.
- Cluster nodes do not have public IP addresses by default.
- Public IP addresses can be assigned to static and master nodes:
  - In this case, One-to-One NAT is used to map the public IP address to the node’s internal IP address (Cloud NAT will not be used in this scenario).
- If the master node does not have a public IP address, an additional instance with a public IP address (for example, a bastion host) is required for cluster installation and access.
- Peering can be configured between the cluster’s VPC and other VPCs.

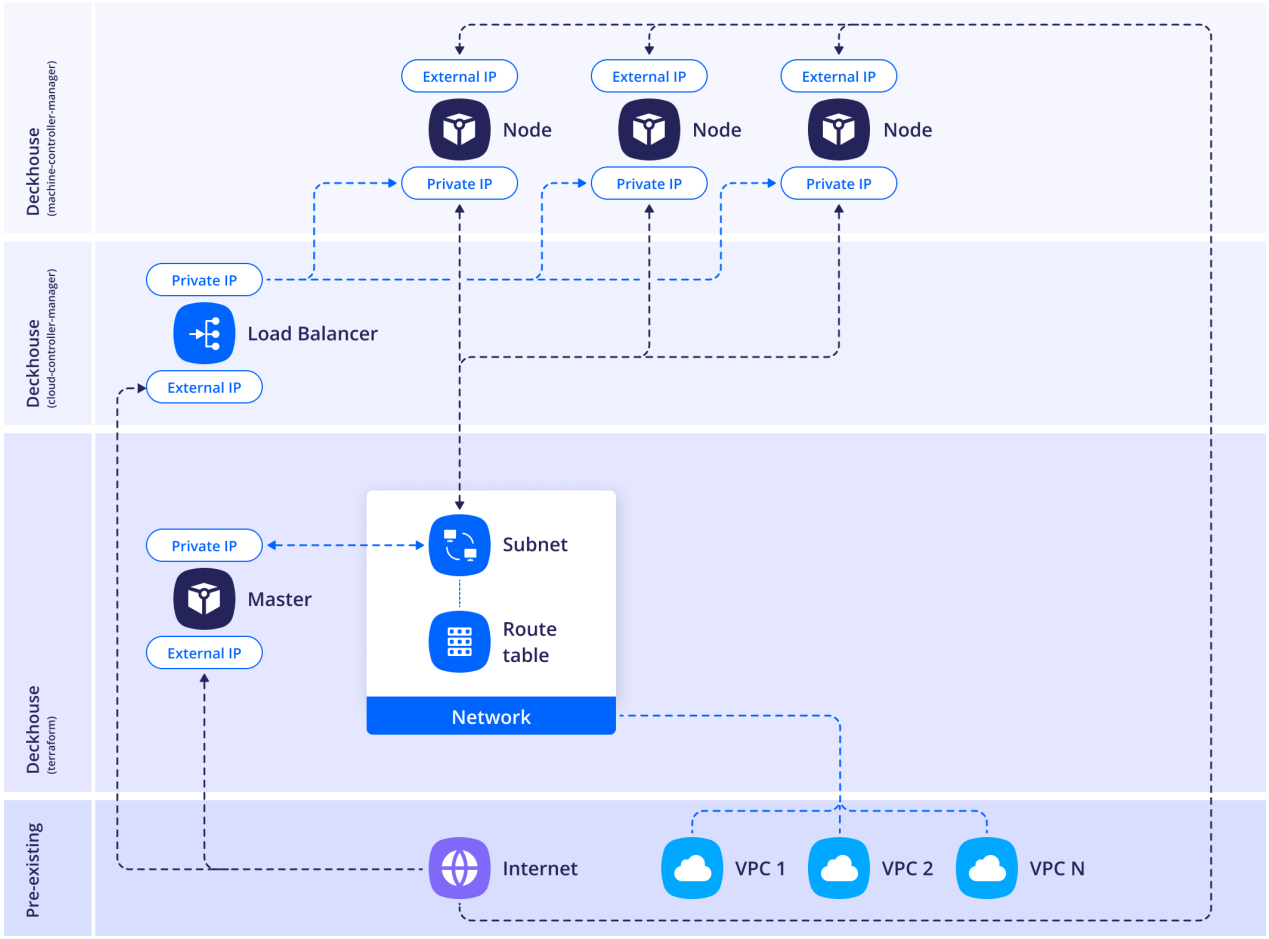


Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: Standard
standard:
  # Optional. Listed addresses will be used for
  # addressing Cloud NAT.
  cloudNATAddresses:
  - example-address-1
  - example-address-2
subnetworkCIDR: 10.0.0.0/24       # Required.
# Optional. A list of GCP VPC Networks used by Kubernetes VPC
# Network to connect over peering.
peeredVPCs:
- default
sshKey: "<SSH_PUBLIC_KEY>"  # Required.
labels:
  kube: example
masterNodeGroup:
  replicas: 1
  zones:                      # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128  # Required.
    diskSizeGb: 20               # Optional. A local disk is used if not set.
    disableExternalIP: false       # Optional. The master node has externalIP by default.
    additionalNetworkTags:        # Optional.
    - tag1
    additionalLabels:            # Optional.
      kube-node: master
nodeGroups:
- name: static
  replicas: 1
  zones:                      # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128  # Required.
    diskSizeGb: 20               # Optional. A local disk is used if not set.
    disableExternalIP: true       # Optional. The nodes don't have externalIP by default.
    additionalNetworkTags:        # Optional.
    - tag1
    additionalLabels:            # Optional.
      kube-node: static
provider:
  region: europe-west4           # Required.
  serviceAccountJSON: |          # Required.
    {
      "type": "service_account",
      "project_id": "sandbox",
      "private_key_id": "98sdcj5e8c7asd98j4j3n9csakn",
      "private_key": "-----BEGIN PRIVATE KEY-----",
      "client_id": "342975658279248",
      "auth_uri": "https://accounts.google.com/o/oauth2/auth",
      "token_uri": "https://oauth2.googleapis.com/token",
      "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
      "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/k8s-
test%40sandbox.iam.gserviceaccount.com"
    }
```

WithoutNAT

- A dedicated VPC is created for the cluster. All cluster nodes are assigned public IP addresses.
- Peering can be configured between the cluster’s VPC and other VPCs.



Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
subnetworkCIDR: 10.0.0.0/24       # Required.
# Optional. A list of GCP VPC Networks used by Kubernetes VPC
# Network to connect over peering.
peeredVPCs:
- default
labels:
  kube: example
masterNodeGroup:
  replicas: 1
  zones:                        # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128  # Required.
    diskSizeGb: 20               # Optional. A local disk is used if not set.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:             # Optional.
      kube-node: master
nodeGroups:
- name: static
  replicas: 1
  zones:                        # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128  # Required.
    diskSizeGb: 20               # Optional. A local disk is used if not set.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:             # Optional.
      kube-node: static
provider:
  region: europe-west4            # Required.
  serviceAccountJSON: |           # Required.
    {
      "type": "service_account",
      "project_id": "sandbox",
      "private_key_id": "98sdcj5e8c7asd98j4j3n9csakn",
      "private_key": "-----BEGIN PRIVATE KEY-----",
      "client_id": "342975658279248",
      "auth_uri": "https://accounts.google.com/o/oauth2/auth",
      "token_uri": "https://oauth2.googleapis.com/token",
      "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
      "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/k8s-
test%40sandbox.iam.gserviceaccount.com"
    }
```

Configuration

Integration with GCP is handled via the GCPClusterConfiguration resource, which describes the cloud cluster configuration in GCP and is used by the cloud provider when the control plane is hosted in the cloud. The responsible DKP module configures itself automatically based on the selected layout.

To update the configuration of a running cluster, run the following command:

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

> **i** After modifying node-related parameters, you must run the `dhctl converge` command for the changes to take effect.

Example configuration:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
sshKey: "<SSH_PUBLIC_KEY>"
subnetworkCIDR: 10.36.0.0/24
masterNodeGroup:
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20240523a
    diskSizeGb: 50
nodeGroups:
- name: static
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20240523a
    diskSizeGb: 50
    additionalNetworkTags:
    - tag1
    additionalLabels:
      kube-node: static
provider:
  region: europe-west3
  serviceAccountJSON: "<SERVICE_ACCOUNT_JSON>"
```

Machine provisioning and parameters are configured in the NodeGroup custom resource, where the instance class for the node group is specified (the `cloudInstances.classReference` parameter). For GCP, the instance class is a GCPInstanceClass custom resource that defines the machine parameters.

DKP also automatically creates StorageClasses that cover all available disk types in GCP:

Disk type	Replication	StorageClass name
standard	none	`pd-standard-not-replicated`
standard	regional	`pd-standard-replicated`
balanced	none	`pd-balanced-not-replicated`
balanced	regional	`pd-balanced-replicated`
ssd	none	`pd-ssd-not-replicated`
ssd	regional	`pd-ssd-replicated`

You can exclude unnecessary StorageClasses by specifying them in the `exclude` parameter.

Enabling nested virtualization

To run virtual machine workloads (e.g., KVM-based VMs) inside GCP instances, enable nested virtualization.

> **i** Only specific machine types support nested virtualization. See the GCP documentation for the list of compatible types.

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: vm-nodes
spec:
  machineType: n2-standard-8
  nestedVirtualization: true
```

Adding additional disks

To attach additional disks to instances (for example, for LINSTOR, Ceph, NFS storage nodes, and similar solutions), specify them in the `additionalDisks` parameter:

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: storage-nodes
spec:
  machineType: n1-standard-8
  additionalDisks:
  - sizeGb: 200
    type: pd-ssd
  - sizeGb: 500
    type: pd-standard
    autoDelete: true
```

Configuring node security policies

You may need to restrict or allow incoming and outgoing traffic on GCP virtual machines for various reasons:

- Allowing access to cluster nodes from VMs in other subnets.
- Allowing access to static node ports for an application.
- Restricting access to external resources or other VMs in the cloud following the requirements of the security team.

To implement this, use additional network tags.

Setting additional network tags for static and master nodes

This parameter can be set during cluster creation or in an existing cluster. In both cases, the additional network tags must be specified in GCPClusterConfiguration:

- **For master nodes**: Under the `additionalNetworkTags` field of the `masterNodeGroup` section.
- **For static nodes**: Under the `additionalNetworkTags` field of the `nodeGroups` section for the corresponding node group.

The `additionalNetworkTags` field is an array of strings with network tag names.

Setting additional network tags for ephemeral nodes

Specify the `additionalNetworkTags` parameter in each GCPInstanceClass resource in the cluster that requires extra network tags.

Adding CloudStatic nodes to the cluster

To add manually created virtual machines to the cluster as nodes, add a `Network Tag` matching the cluster prefix.

You can find the cluster prefix by running the following command:

```
kubectl -n kube-system get secret d8-cluster-configuration -o json | jq -r '.data."cluster-configuration.yaml"' \
  | base64 -d | grep prefix
```

Additional resources

- Module `cloud-provider-gcp` provider configuration
- Module `cloud-provider-gcp` custom resources
- Module `cloud-provider-gcp` parameters
- Module `node-manager` custom resources

Private cloud providers

Deckhouse Kubernetes Platform (DKP) provides built-in tools for integration with private clouds deployed in your own infrastructure. These tools let you automatically provision virtual machines and connect them to a cluster, deploy clusters within private cloud environments, and scale resources as needed.

- Deckhouse Virtualization Platform (DVP)
- Huawei Cloud
- OpenStack
- VK Private Cloud

The following sections describe how you can configure integration with each of these private cloud platforms.

Connection and authorization in VK Cloud

To manage resources in VK Private Cloud using Deckhouse Kubernetes Platform, you must connect to the VK Private Cloud API.

The list of VK Private Cloud API services required for deployment is available in the Configuration section.

The user credentials required to access the VK Private Cloud API are provided in the OpenStack RC file (openrc file).

Instructions for downloading the openrc file via the standard web interface of VK Private Cloud and using it are available in the VK Private Cloud documentation.

If you’re using a cloud provider’s implementation of the VK Private Cloud API, the process for obtaining the openrc file may

differ.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Layouts and configuration in VK Cloud

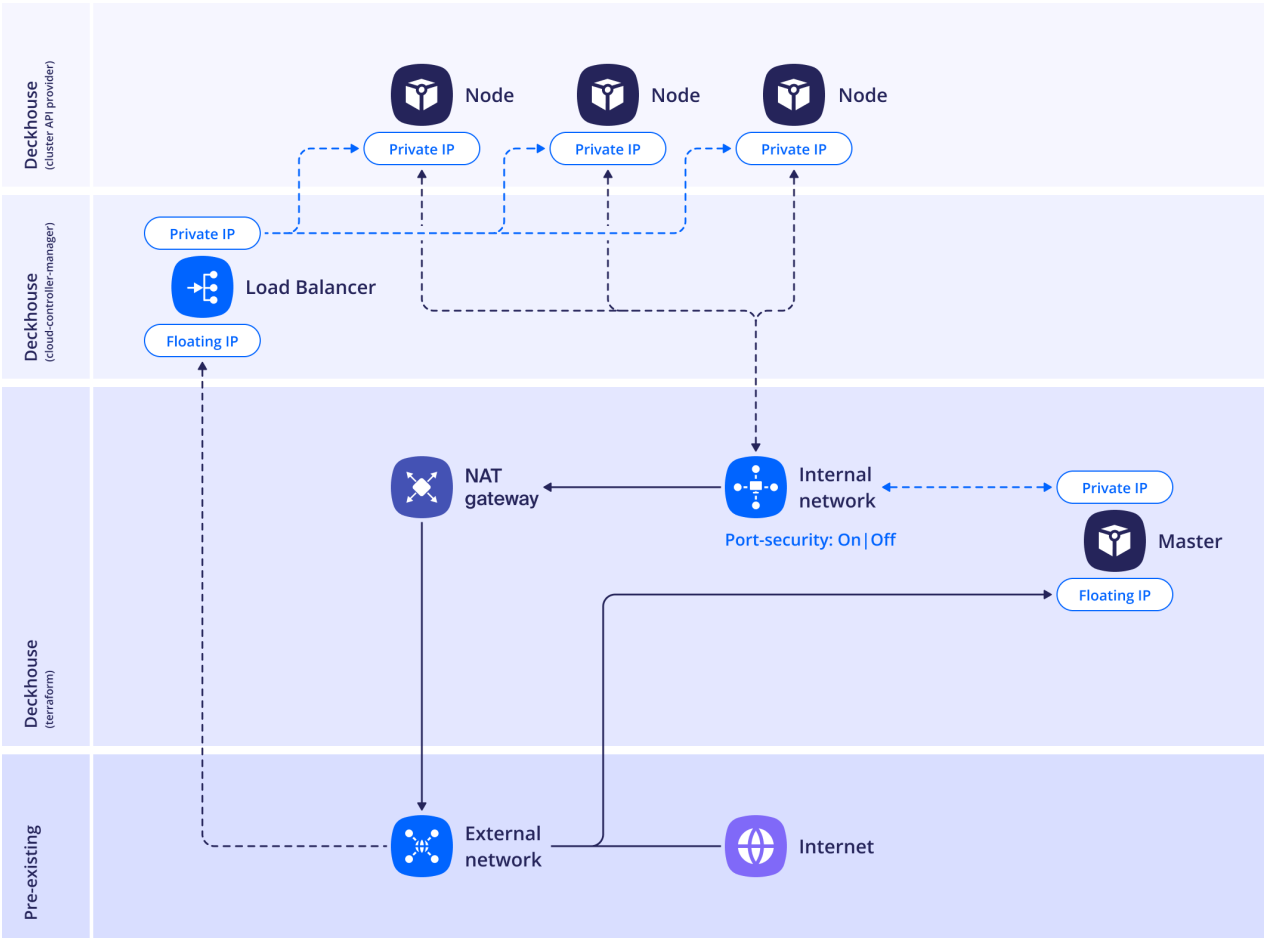
Layouts

This section describes the possible node placement layouts in VK Private Cloud infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. The floating IP is assigned to the master node.

> ⚠ If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port `6443`. To avoid this, we recommend using the SimpleWithInternalNetwork or Standard layout with a bastion host.



Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                 # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, true by default.
  externalNetworkName: shared                # Required.
  bastion:
    zone: ru2-b                             # Optional.
    volumeType: fast-ru-2b                   # Optional.
    instanceClass:
      flavorName: m1.large                   # Required.
      imageName: ubuntu-20-04-cloud-amd64     # Required.
      rootDiskSize: 50                      # Optional, 50 GB by default.
      additionalTags:
        severity: critical                   # Optional.
        environment: production              # Optional.
```

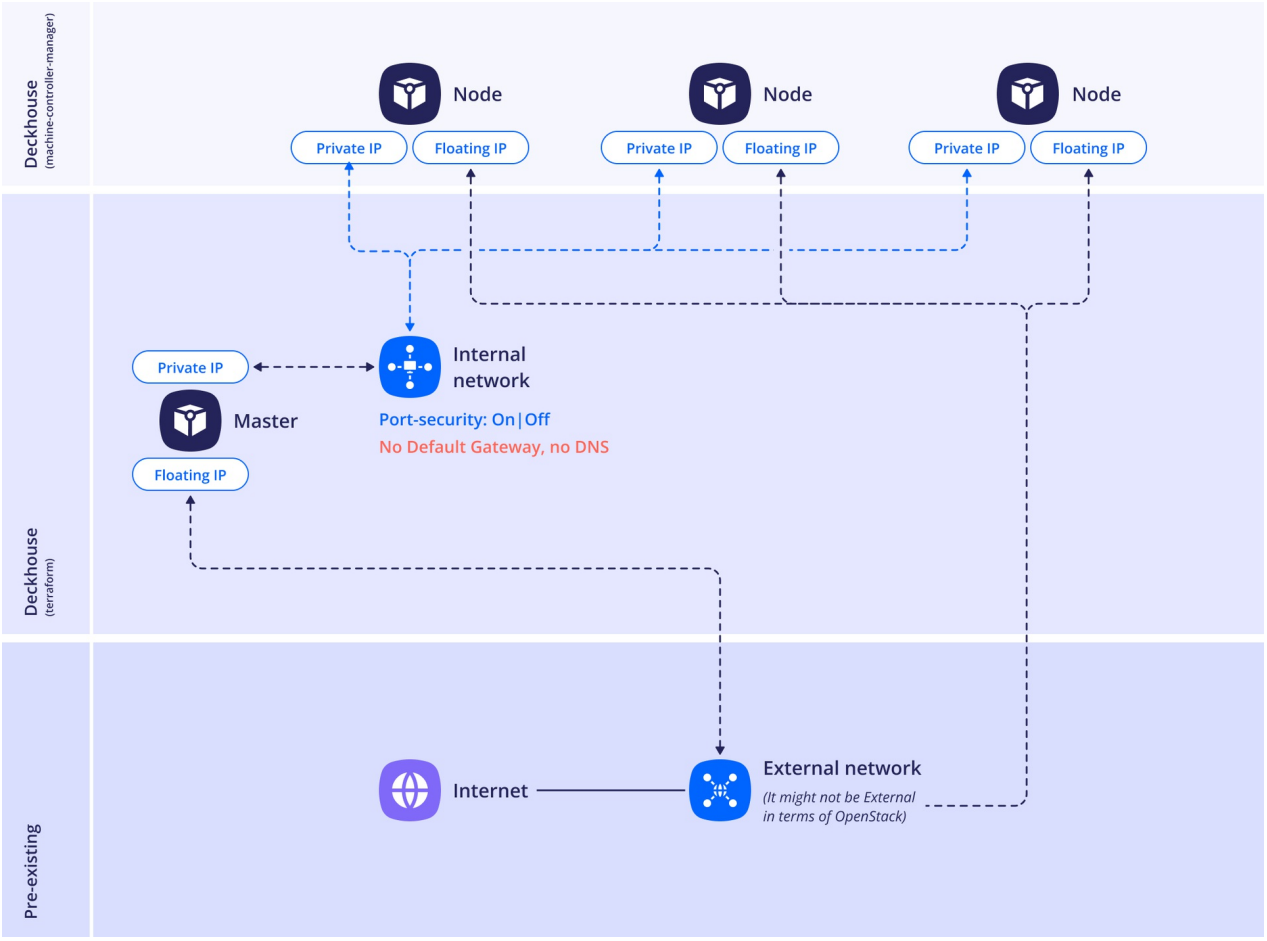
```
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default. Determines if configuration drive is
    # required during vm bootstrap process. It's needed if there
    # is no DHCP in the network that is used as a default gateway.
    configDrive: false
    # Required, the gateway of this network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

⚠

- This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.
- Using this strategy, make sure to explicitly specify the name of the internal network in `additionalNetworks` when creating an OpenStackInstanceClass in the cluster.



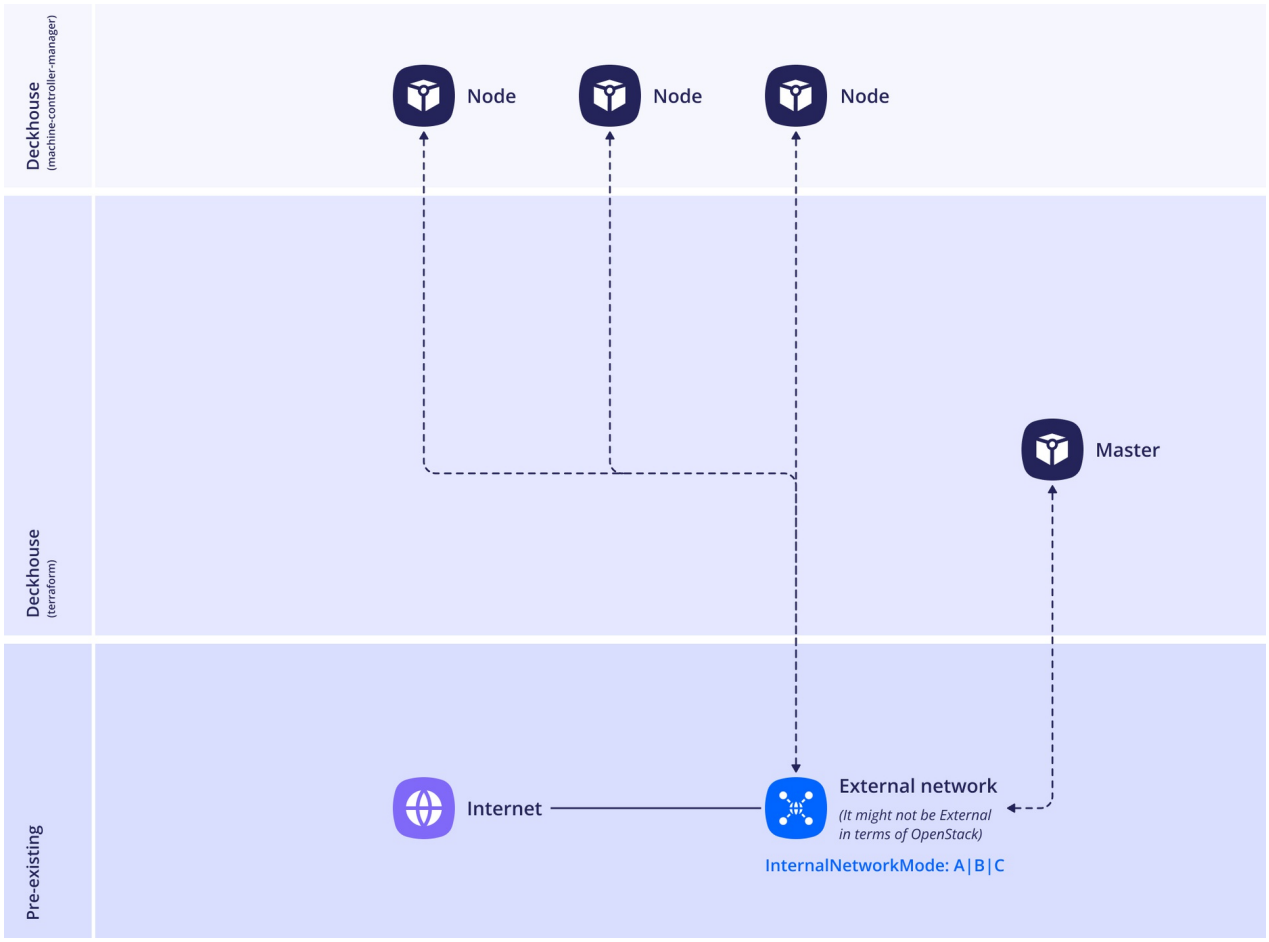
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, defines whether DHCP is enabled in specified external
  # network (true by default).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, true by default.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy can be useful if you need to merge a Kubernetes cluster with existing VMs.

⚠ This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.



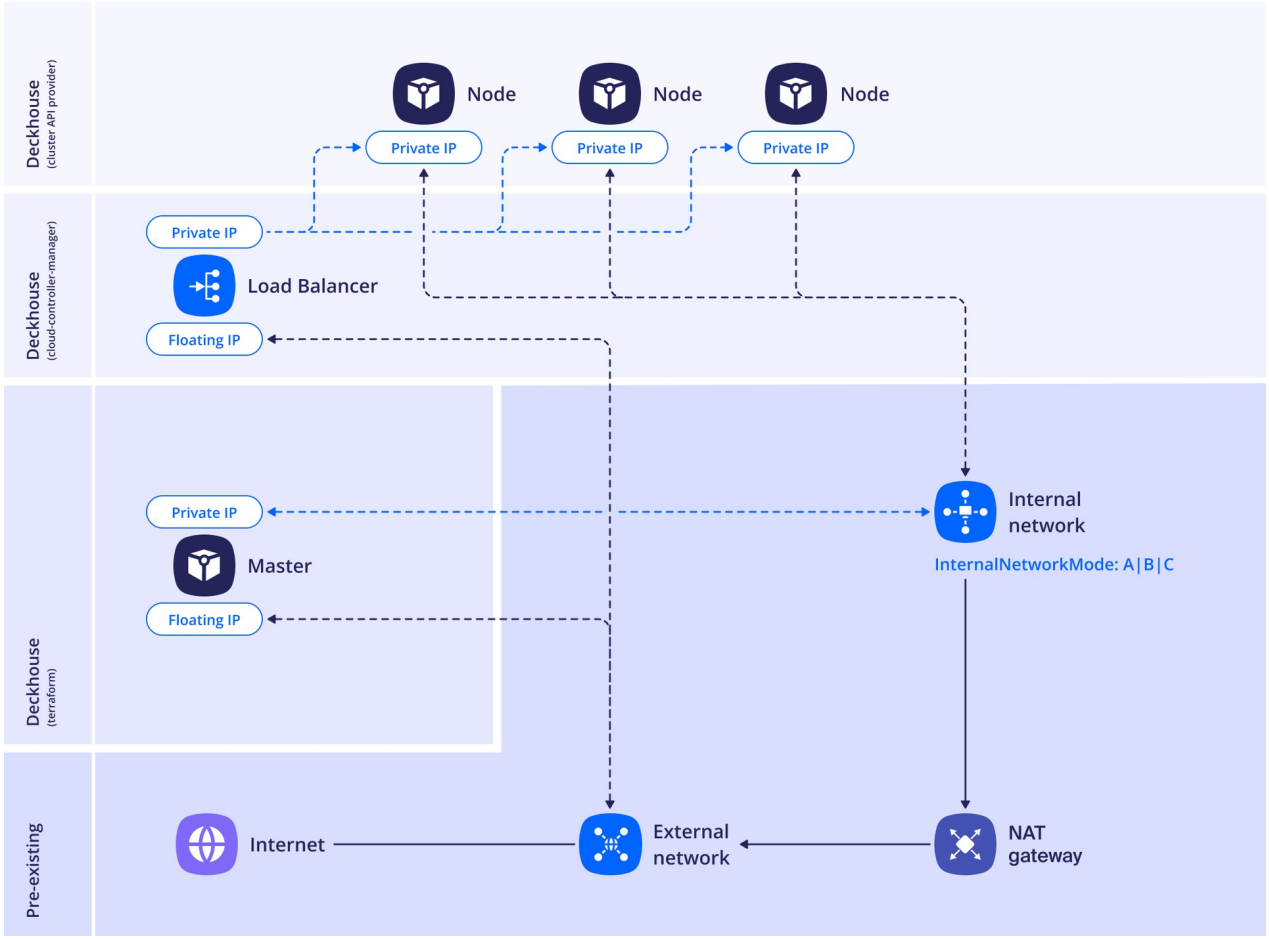
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, true by default.
  # Optional, VXLAN by default, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This layout can be useful if you need to merge a Kubernetes cluster with existing VMs.

> ⚠ This layout does not involve the management of SecurityGroups (it is assumed they were created beforehand). To configure security policies, you must explicitly specify both `additionalSecurityGroups` in the OpenStackClusterConfiguration for the masterNodeGroup and other nodeGroups, and `additionalSecurityGroups` when creating OpenStackInstanceClass in the cluster.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Required, all cluster nodes have to be in the same subnet.
  internalSubnetName: pivot-standard
  # Optional, DirectRoutingWithPortSecurityEnabled by default,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional. If set, it will be used for the load balancer default
  # configuration and ordering the master floating IP.
  externalNetworkName: ext-net
  # Optional, true by default.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Configuration

Integration with VK Private Cloud is performed using the OpenStackClusterConfiguration resource. It defines the configuration of the cloud cluster in VK Private Cloud and is used by the cloud provider when the cluster control plane is hosted in the cloud. The DKP module responsible for the integration is automatically configured based on the selected layout.

To modify the configuration in a running cluster, run the following command:

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

> **i** After changing node-related parameters, run the `dhctl converge` command to apply the changes.

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the VK Private Cloud cloud provider, the instance class is a custom resource called OpenStackInstanceClass, which contains the specific configuration of the VMs.

> **i** An OpenStack cluster is deployed in a single region (`provider.region`). Zones from other regions cannot be used. For details on configuring zones for CloudPermanent and CloudEphemeral nodes, see How do I create NodeGroups in availability zones? section.

> ⚠ When the module settings are changed, **existing Machine objects in the cluster are NOT recreated** (new Machine objects will use the updated parameters). The recreation only occurs if you modify the NodeGroup or the OpenStackInstanceClass parameters.

Configuration examples

The following are two simple configuration examples for the OpenStack cloud provider.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

List of required services

Below is the list of VK Private Cloud services required for DKP to operate in VK Private Cloud:

Service	API version
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* If you need to provision a LoadBalancer.

For the API endpoints and ports, refer to the official documentation.

LoadBalancer configuration

⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

It is recommended to use `loadbalancer.openstack.org/node-selector` to select only the nodes that should be used as targets for the corresponding LoadBalancer.

IngressNginxController example

In this example, the Ingress controller pods are scheduled on frontend nodes, while the `loadbalancer.openstack.org/node-selector` annotation limits the load balancer pool to the same nodes:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Node security policies and configuration

There are many reasons why you might want to restrict or allow incoming or outgoing traffic on cluster VMs. For example:

- Allow connections to cluster nodes from VMs in a different subnet.
- Allow access to specific ports on a static node for application traffic.
- Restrict access to external resources or other VMs in the cloud per security team requirements.

To manage this, you should use additional security groups. Only security groups that were pre-created in the cloud can be used.

Assigning additional security groups to static and master nodes

You can specify additional security groups either during cluster creation or in an existing cluster. In both cases, specify them in the OpenStackClusterConfiguration resource:

- **For master nodes**: Under the `masterNodeGroup` section using the `additionalSecurityGroups` field.
- **For static nodes**: Under the `nodeGroups` section of the corresponding node group configuration and also in the `additionalSecurityGroups` field.

The `additionalSecurityGroups` field is an array of strings representing the names of the security groups.

Assigning additional security groups to ephemeral nodes

To assign additional security groups to ephemeral nodes, specify the `additionalSecurityGroups` parameter in all relevant OpenStackInstanceClass resources used by these nodes.

Uploading an image to VK Private Cloud

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an openrc file that contains credentials for accessing the VK Private Cloud API.

   > **i** The method of obtaining the openrc file may vary depending on the VK Private Cloud provider. If your provider offers the standard interface for VK Private Cloud, follow this guide to download the file.

3. Alternatively, install the OpenStack CLI tools following the instructions.

   You can also run a container, mounting both the openrc file and the downloaded Ubuntu image:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize environment variables from the openrc file:

   ```
   source /openrc
   ```

5. List the available volume types:

   ```
   openstack volume type list
   ```

   Example output:

   ```
   +--------------------------------------+---------------+-----------+
   | ID                                   | Name          | Is Public |
   +--------------------------------------+---------------+-----------+
   | 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
   | bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
   | 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
   | eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
   | 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
   | a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
   | ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
   | a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
   +--------------------------------------+---------------+-----------+
   ```

6. Create an image and assign the volume type as a property (this is necessary if VK Private Cloud does not support local disks or they are unsuitable for workloads):

   ```
   openstack image create --private --disk-format qcow2 --container-format bare \
     --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
   ```

7. Ensure the image has been created:

   ```
   openstack image show ubuntu-18-04-cloud-amd64
   ```

   Example output:

```
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| Field            | Value
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| checksum         | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-
4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url':
u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap',
u'metadata': {}}]' |
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
```

Verifying security group support by the cloud provider

Run the `openstack security group list` command. If you don’t receive any errors, it means that security groups are supported.

Configuring online volume resize

When resizing a disk via the OpenStack API, Cinder does not pass updated size information to Nova. As a result, the disk size inside the guest OS remains unchanged.

To fix this, you must configure access to the Nova API in the `cinder.conf` file. For example:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

For more details, see the OpenStack-Ansible bug report.

Using rootDiskSize

Disks in VK Private Cloud

A node's disk can be either local or network-attached. In VK Private Cloud terminology:

- A local disk is called an ephemeral disk.
- A network-attached disk is called a persistent disk (Cinder storage).

The ephemeral disk is deleted along with the VM, while the persistent disk remains in the cloud even after the VM is deleted.

- For master nodes, network-attached disks are preferred so the nodes could migrate between hypervisors.
- For ephemeral nodes, local disks are preferred for cost efficiency. However, not all cloud providers support local disks. If not supported, you have to use network-attached disks for ephemeral nodes.

Local disk (ephemeral)	Network-attached disk (persistent)
Deleted with the VM	Remains in cloud and can be reused
Less expensive	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

rootDiskSize parameter

The OpenStackInstanceClass resource includes a `rootDiskSize` parameter. In VK Private Cloud, flavors may also define a disk size.

The resulting disk type depends on the combination of the following parameters:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` not set	*Disk size must be specified*. VM creation will fail without it.	Local disk with size from the flavor.
`rootDiskSize` set	Network-attached disk with size that equals `rootDiskSize`.	Both network-attached (`rootDiskSize`) and local (from flavor). Avoid this setup, as the cloud provider may charge for both.

> **i** In Selectel Cloud, when creating nodes with CloudEphemeral type outside zone A, you must pre-create a flavor with the target disk size. In this case, you do not need to set the `rootDiskSize` parameter.

Recommendations for master nodes and bastion with network-attached disks

- Use a flavor with zero disk size.
- Set `rootDiskSize` in the OpenStackInstanceClass resource.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the type will be taken from `volumeTypeMap`.

Recommendations for ephemeral nodes with local disks

- Use a flavor with a defined disk size.
- Do not use the `rootDiskSize` parameter in OpenStackInstanceClass.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the cloud provider's default disk type will be used.

Checking the disk size in a flavor

Run the following command:

```
openstack flavor show m1.medium-50g -C disk
```

Example output:

```
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

Overriding the cloud provider’s default disk type

If your cloud provider offers multiple volume types, you can specify the default type for an image by setting the `cinder_img_volume_type` property in the image metadata. Any VM created from that image will use the specified volume type.

You can also create a new VK Private Cloud image and upload it.

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Connection and authorization in OpenStack

To manage resources in OpenStack using Deckhouse Kubernetes Platform, you must connect to the OpenStack API.

The list of OpenStack API services required for deployment is available in the Configuration section.

The user credentials required to access the OpenStack API are provided in the OpenStack RC file (openrc file).

Instructions for downloading the openrc file via the standard web interface of OpenStack and using it are available in the OpenStack documentation.

If you’re using a cloud provider’s implementation of the OpenStack API, the process for obtaining the openrc file may differ.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Layouts and configuration in OpenStack

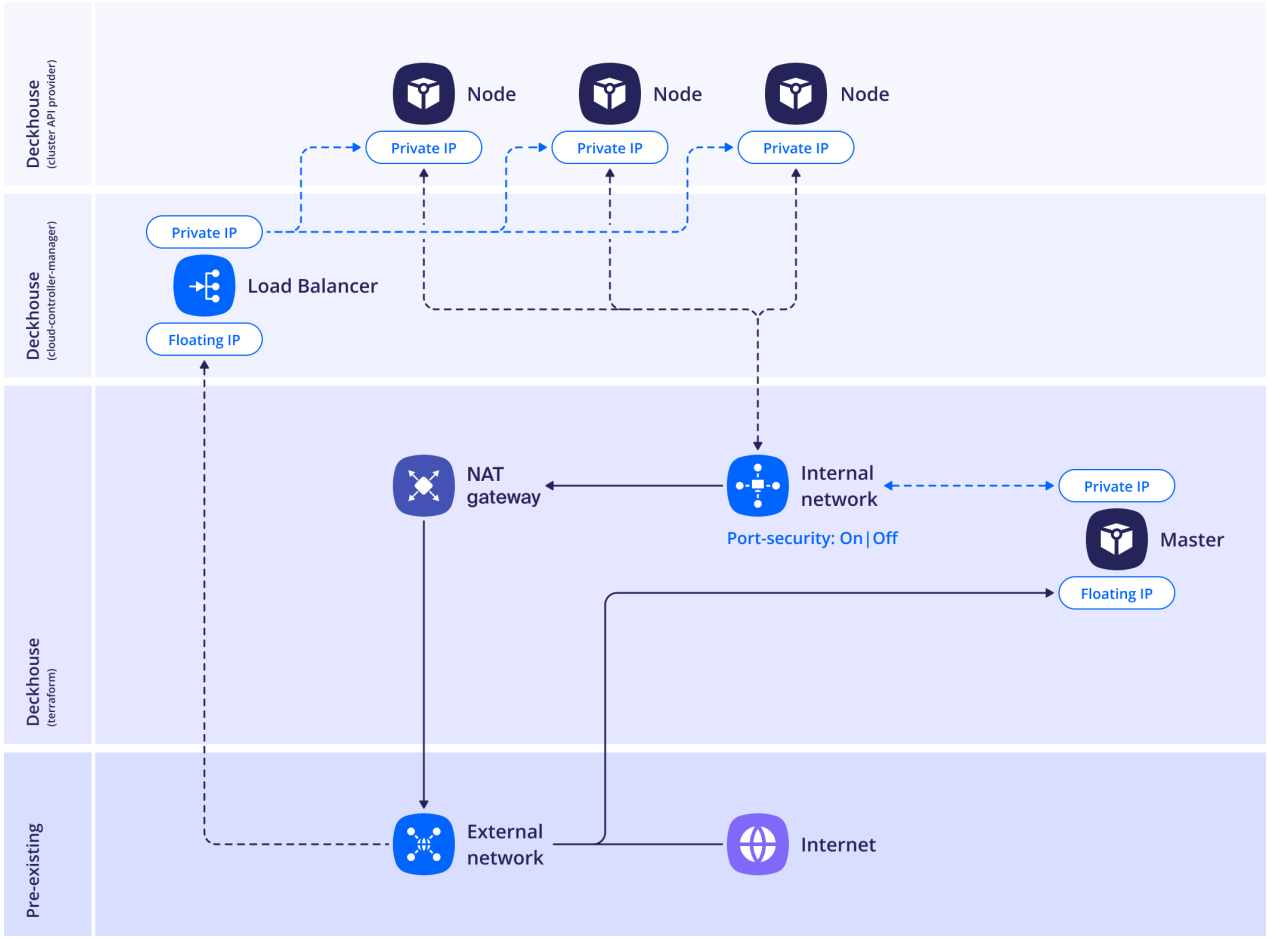
Layouts

This section describes the possible node placement layouts in OpenStack infrastructure and the related configuration options. The selected layout affects networking behavior, availability of public IP addresses, outgoing traffic routing, and how nodes are accessed.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. The floating IP is assigned to the master node.

⚠ If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port `6443`. To avoid this, we recommend using the SimpleWithInternalNetwork or Standard layout with a bastion host.



Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                 # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, true by default.
  externalNetworkName: shared                # Required.
  bastion:
    zone: ru2-b                             # Optional.
    volumeType: fast-ru-2b                   # Optional.
    instanceClass:
      flavorName: m1.large                   # Required.
      imageName: ubuntu-20-04-cloud-amd64      # Required.
      rootDiskSize: 50                      # Optional, 50 GB by default.
      additionalTags:
        severity: critical                   # Optional.
        environment: production              # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
```

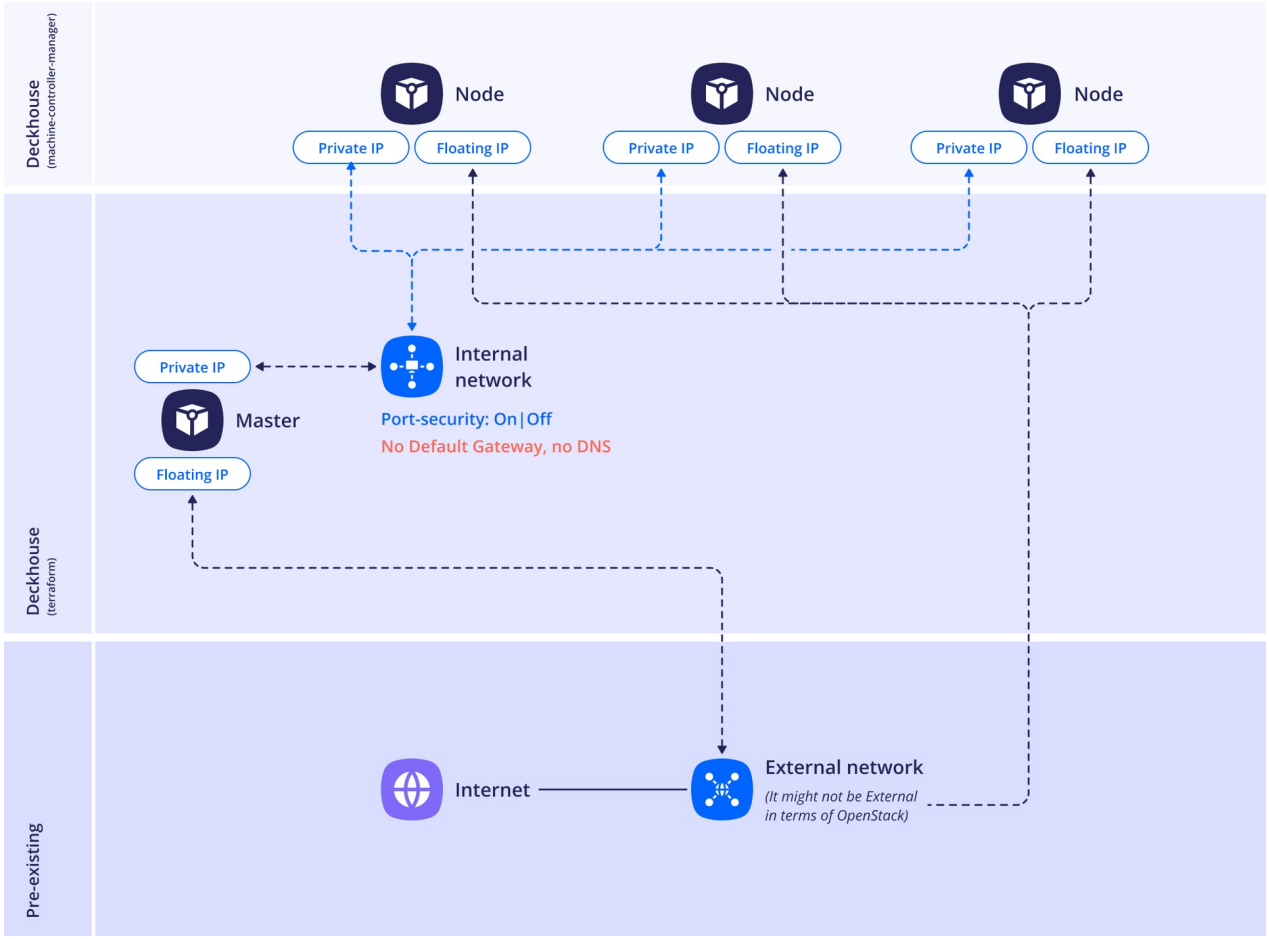
```
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                  # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default. Determines if configuration drive is
    # required during vm bootstrap process. It's needed if there
    # is no DHCP in the network that is used as a default gateway.
    configDrive: false
    # Required, the gateway of this network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                   # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

⚠

- This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.
- Using this strategy, make sure to explicitly specify the name of the internal network in `additionalNetworks` when creating an OpenStackInstanceClass in the cluster.



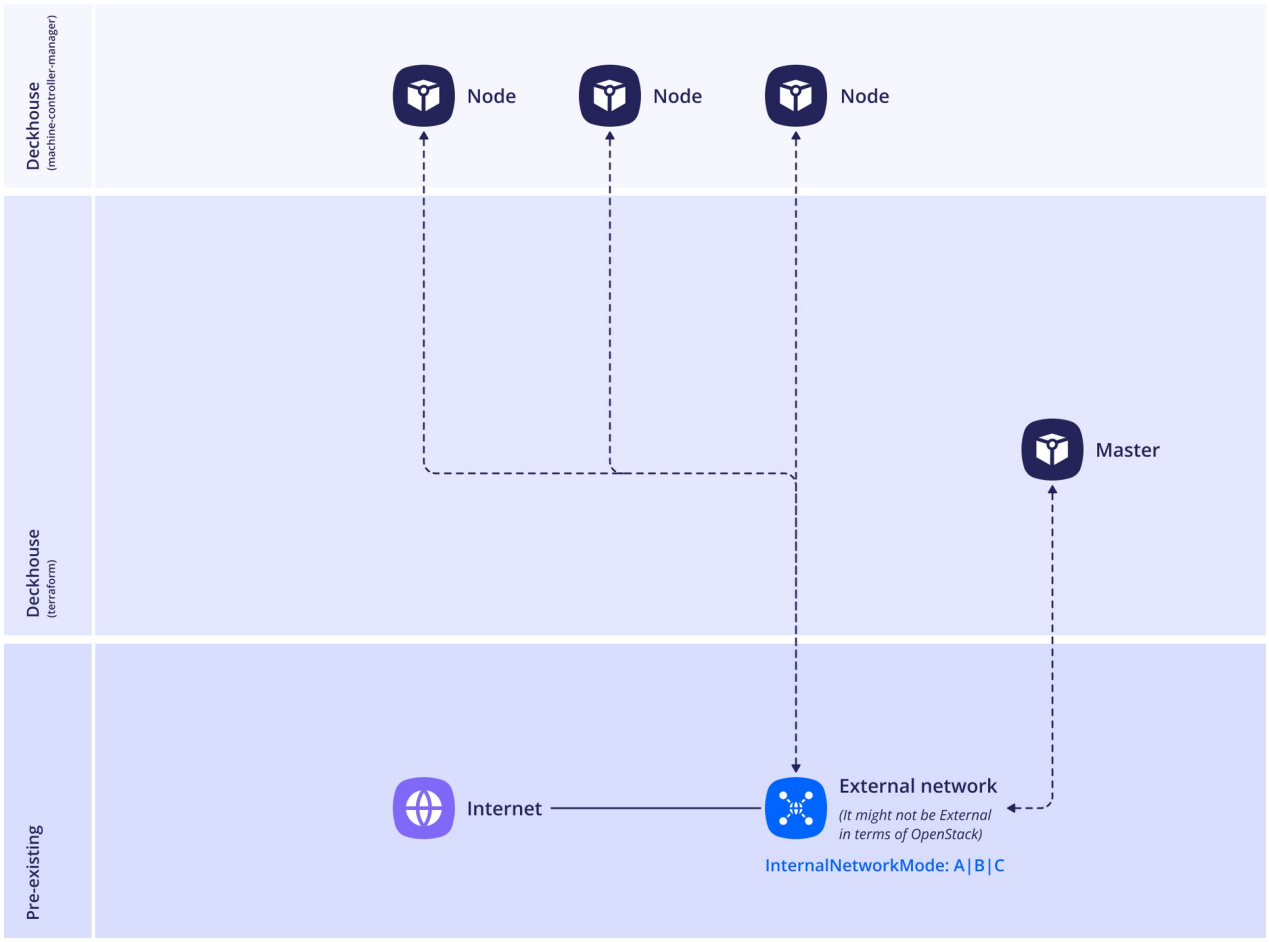
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, defines whether DHCP is enabled in specified external
  # network (true by default).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, true by default.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume.
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy can be useful if you need to merge a Kubernetes cluster with existing VMs.

> ⚠ This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a LoadBalancer with the Floating IP. An internal LoadBalancer with the virtual IP in the public network is only accessible from cluster nodes.



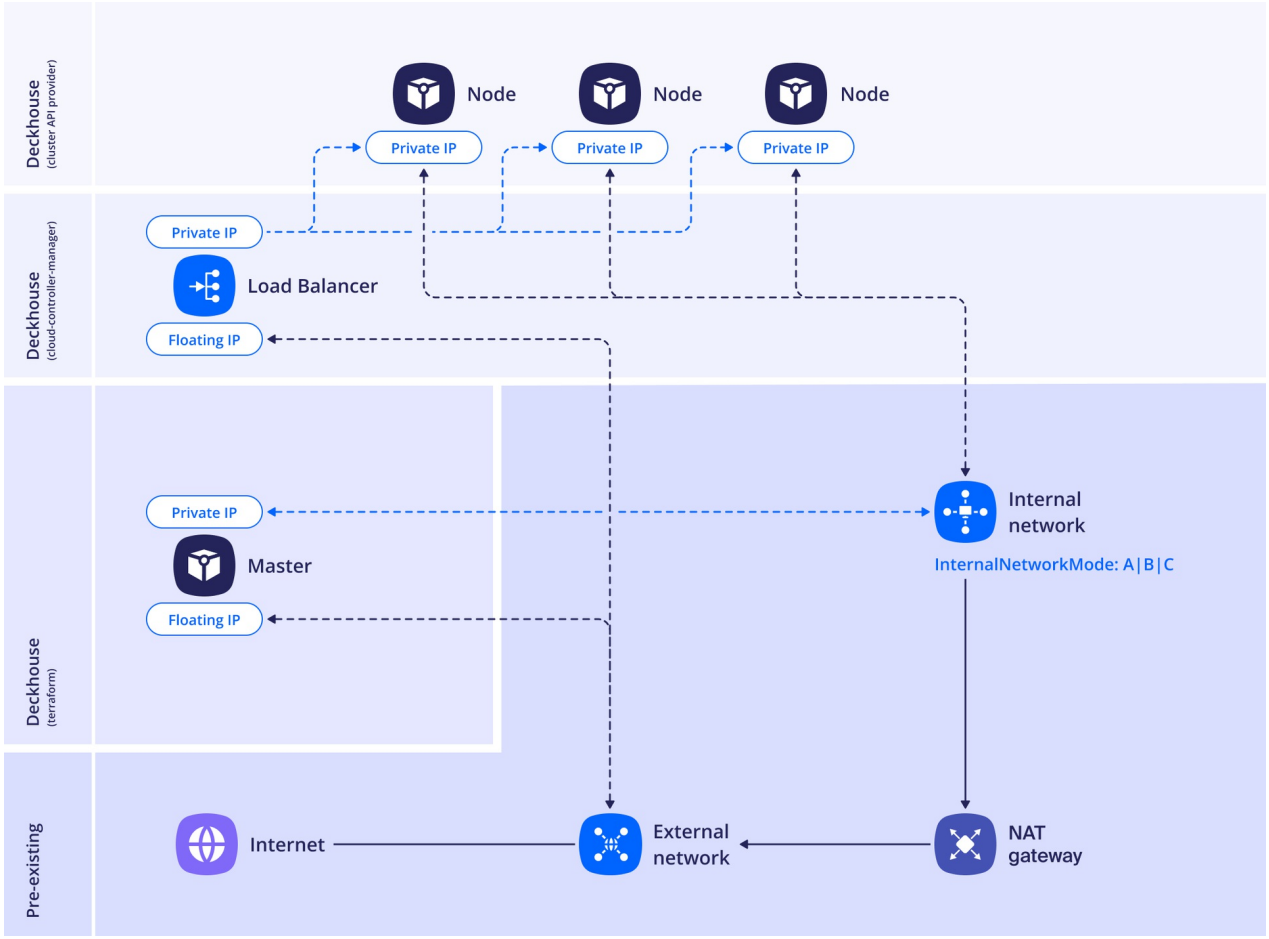
Example layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, true by default.
  # Optional, VXLAN by default, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This layout can be useful if you need to merge a Kubernetes cluster with existing VMs.

> ⚠ This layout does not involve the management of SecurityGroups (it is assumed they were created beforehand). To configure security policies, you must explicitly specify both `additionalSecurityGroups` in the OpenStackClusterConfiguration for the masterNodeGroup and other nodeGroups, and `additionalSecurityGroups` when creating OpenStackInstanceClass in the cluster.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Required, all cluster nodes have to be in the same subnet.
  internalSubnetName: pivot-standard
  # Optional, DirectRoutingWithPortSecurityEnabled by default,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional. If set, it will be used for the load balancer default
  # configuration and ordering the master floating IP.
  externalNetworkName: ext-net
  # Optional, true by default.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and Kubernetes certificates
  # (always use the fastest disk supplied by the provider).
  volumeTypeMap:
    # If rootDiskSize is specified, this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, false by default, determines if configuration drive
    # is required during vm bootstrap process. It's needed
    # if there is no DHCP in the network that is used as a default
    # gateway.
    configDrive: false
    # Required, the network will be used as a default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security, their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, a list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Configuration

Integration with OpenStack is performed using the OpenStackClusterConfiguration resource. It defines the configuration of the cloud cluster in OpenStack and is used by the cloud provider when the cluster control plane is hosted in the cloud. The DKP module responsible for the integration is automatically configured based on the selected layout.

To modify the configuration in a running cluster, run the following command:

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

> **i** After changing node-related parameters, run the `dhctl converge` command to apply the changes.

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the OpenStack cloud provider, the instance class is a custom resource called OpenStackInstanceClass, which contains the specific configuration of the VMs.

> **i** An OpenStack cluster is deployed in a single region (`provider.region`). Zones from other regions cannot be used. For details on configuring zones for CloudPermanent and CloudEphemeral nodes, see How do I create NodeGroups in availability zones? section.

> ⚠ When the module settings are changed, **existing Machine objects in the cluster are NOT recreated** (new Machine objects will use the updated parameters). The recreation only occurs if you modify the NodeGroup or the OpenStackInstanceClass parameters.

Configuration examples

The following are two simple configuration examples for the OpenStack cloud provider.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

List of required services

Below is the list of OpenStack services required for DKP to operate in OpenStack:

Service	API version
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* If you need to provision a LoadBalancer.

LoadBalancer configuration

⚠ To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

It is recommended to use `loadbalancer.openstack.org/node-selector` to select only the nodes that should be used as targets for the corresponding LoadBalancer.

IngressNginxController example

In this example, the Ingress controller pods are scheduled on frontend nodes, while the `loadbalancer.openstack.org/node-selector` annotation limits the load balancer pool to the same nodes:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Node security policies and configuration

There are many reasons why you might want to restrict or allow incoming or outgoing traffic on cluster VMs. For example:

- Allow connections to cluster nodes from VMs in a different subnet.
- Allow access to specific ports on a static node for application traffic.
- Restrict access to external resources or other VMs in the cloud per security team requirements.

To manage this, you should use additional security groups. Only security groups that were pre-created in the cloud can be used.

Assigning additional security groups to static and master nodes

You can specify additional security groups either during cluster creation or in an existing cluster. In both cases, specify them in the OpenStackClusterConfiguration resource:

- **For master nodes**: Under the `masterNodeGroup` section using the `additionalSecurityGroups` field.
- **For static nodes**: Under the `nodeGroups` section of the corresponding node group configuration and also in the `additionalSecurityGroups` field.

The `additionalSecurityGroups` field is an array of strings representing the names of the security groups.

Assigning additional security groups to ephemeral nodes

To assign additional security groups to ephemeral nodes, specify the `additionalSecurityGroups` parameter in all

relevant OpenStackInstanceClass resources used by these nodes.

Uploading an image to OpenStack

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an openrc file that contains credentials for accessing the OpenStack API.

   > **i** The method of obtaining the openrc file may vary depending on the OpenStack provider. If your provider offers the standard interface for OpenStack, follow this guide to download the file.

3. Alternatively, install the OpenStack CLI tools following the instructions.

   You can also run a container, mounting both the openrc file and the downloaded Ubuntu image:

   ```
   docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
   ```

4. Initialize environment variables from the openrc file:

   ```
   source /openrc
   ```

5. List the available volume types:

   ```
   openstack volume type list
   ```

   Example output:

   ```
   +--------------------------------------+---------------+-----------+
   | ID                                   | Name          | Is Public |
   +--------------------------------------+---------------+-----------+
   | 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
   | bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
   | 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
   | eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
   | 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
   | a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
   | ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
   | a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
   +--------------------------------------+---------------+-----------+
   ```

6. Create an image and assign the volume type as a property (this is necessary if OpenStack does not support local disks or they are unsuitable for workloads):

   ```
   openstack image create --private --disk-format qcow2 --container-format bare \
     --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
   ```

7. Ensure the image has been created:

   ```
   openstack image show ubuntu-18-04-cloud-amd64
   ```

   Example output:

```
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| Field            | Value
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
| checksum         | 3443a1fd810f4af9593d56e0e144d07d
|
| container_format | bare
|
| created_at       | 2020-01-10T07:23:48Z
|
| disk_format      | qcow2
|
| file             | /v2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/file
|
| id               | 01998f40-57cc-4ce3-9642-c8654a6d14fc
|
| min_disk         | 0
|
| min_ram          | 0
|
| name             | ubuntu-18-04-cloud-amd64
|
| owner            | bbf506e3ece54e21b2acf1bf9db4f62c
|
| properties       | cinder_img_volume_type='dp1-high-iops', direct_url='rbd://b0e441fc-c317-
4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap', locations='[{u'url':
u'rbd://b0e441fc-c317-4acf-a606-cf74683978d2/images/01998f40-57cc-4ce3-9642-c8654a6d14fc/snap',
u'metadata': {}}]' |
| protected        | False
|
| schema           | /v2/schemas/image
|
| size             | 343277568
|
| status           | active
|
| tags             |
|
| updated_at       | 2020-05-01T17:18:34Z
|
| virtual_size     | None
|
| visibility       | private
|
+------------------+------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------+
```

Verifying security group support by the cloud provider

Run the `openstack security group list` command. If you don’t receive any errors, it means that security groups are supported.

Configuring online volume resize

When resizing a disk via the OpenStack API, Cinder does not pass updated size information to Nova. As a result, the disk size inside the guest OS remains unchanged.

To fix this, you must configure access to the Nova API in the `cinder.conf` file. For example:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

For more details, see the OpenStack-Ansible bug report.

Using rootDiskSize

Disks in OpenStack

A node's disk can be either local or network-attached. In OpenStack terminology:

- A local disk is called an ephemeral disk.
- A network-attached disk is called a persistent disk (Cinder storage).

The ephemeral disk is deleted along with the VM, while the persistent disk remains in the cloud even after the VM is deleted.

- For master nodes, network-attached disks are preferred so the nodes could migrate between hypervisors.
- For ephemeral nodes, local disks are preferred for cost efficiency. However, not all cloud providers support local disks. If not supported, you have to use network-attached disks for ephemeral nodes.

Local disk (ephemeral)	Network-attached disk (persistent)
Deleted with the VM	Remains in cloud and can be reused
Less expensive	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

rootDiskSize parameter

The OpenStackInstanceClass resource includes a `rootDiskSize` parameter. In OpenStack, flavors may also define a disk size.

The resulting disk type depends on the combination of the following parameters:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` not set	*Disk size must be specified*. VM creation will fail without it.	Local disk with size from the flavor.
`rootDiskSize` set	Network-attached disk with size that equals `rootDiskSize`.	Both network-attached (`rootDiskSize`) and local (from flavor). Avoid this setup, as the cloud provider may charge for both.

> **i** In Selectel Cloud, when creating nodes with CloudEphemeral type outside zone A, you must pre-create a flavor with the target disk size. In this case, you do not need to set the `rootDiskSize` parameter.

Recommendations for master nodes and bastion with network-attached disks

- Use a flavor with zero disk size.
- Set `rootDiskSize` in the OpenStackInstanceClass resource.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the type will be taken from `volumeTypeMap`.

Recommendations for ephemeral nodes with local disks

- Use a flavor with a defined disk size.
- Do not use the `rootDiskSize` parameter in OpenStackInstanceClass.
- Ensure the disk type is correct:
  - If specified in the OS image, it will be used.
  - If not specified, the cloud provider's default disk type will be used.

Checking the disk size in a flavor

Run the following command:

```
openstack flavor show m1.medium-50g -C disk
```

Example output:

```
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

Overriding the cloud provider’s default disk type

If your cloud provider offers multiple volume types, you can specify the default type for an image by setting the `cinder_img_volume_type` property in the image metadata. Any VM created from that image will use the specified volume type.

You can also create a new OpenStack image and upload it.

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

Offline disk resizing

If you encounter the following error during volume resizing, reduce the number of replicas in the affected StatefulSet to 0, wait for the volume size to update, and then scale the replicas back to the original value.

```
Warning  VolumeResizeFailed     5s (x11 over 41s)  external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-
9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

Connection and authorization in Huawei Cloud

Requirements

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

To ensure proper operation of Deckhouse Kubernetes Platform (DKP) with Huawei Cloud, make sure of the following:

- The `cloud-init` package is installed on the virtual machines.
- After the VMs start, the following services must be active:
  - `cloud-config.service`
  - `cloud-final.service`
  - `cloud-init.service`

Accessing the Huawei Cloud API

DKP uses the Huawei Cloud API to manage resources. To configure access, you need to create an IAM user and assign the necessary permissions.

Creating a user group

To create a user group and assign policies, follow these steps:

1. Go to the **Identity and Access Management (IAM)** section.
2. Open the **User Groups** tab and click **Create User Group**.
3. Specify a group name (for example, `deckhouse`) and click **OK**.
4. Select the created group and go to the **Permissions** tab.
5. Click **Authorize** and assign the following policies:
   - `ECS Admin`
   - `VPC Administrator`
   - `NAT Admin`
   - `ELB FullAccess`
   - `DEW KeypairFullAccess`
6. Confirm your selections by clicking **Next**, then **OK**, and finalize with **Finish**.

Creating an IAM user

To create an IAM user, follow these steps:

1. Go to the **Users** tab and click **Create User**.

2. Enter a username (for example, `deckhouse`).
3. Under **Access type**, select **Programmatic access**, and ensure that **Management console access** is disabled.
4. Under **Credential Type**, choose **Access key**.
5. Click **Next**, select the previously created group, and then click **Create**.
6. Download the **Access Key ID** and **Secret Access Key**. These credentials are required to access the Huawei Cloud API and cannot be recovered later.

> **i** Make sure the saved keys are securely stored, as they are required to access the cloud API.

JSON policies

Below are the policy contents in JSON format:

ECS Admin policy

```
{
"Version": "1.1",
"Statement": [
{
    "Action": [
    "ecs:*:*",
    "evs:*:get",
    "evs:*:list",
    "evs:volumes:create",
    "evs:volumes:delete",
    "evs:volumes:attach",
    "evs:volumes:detach",
    "evs:volumes:manage",
    "evs:volumes:update",
    "evs:volumes:use",
    "evs:volumes:uploadImage",
    "evs:snapshots:create",
    "vpc:*:get",
    "vpc:*:list",
    "vpc:networks:create",
    "vpc:networks:update",
    "vpc:subnets:update",
    "vpc:subnets:create",
    "vpc:ports:*",
    "vpc:routers:get",
    "vpc:routers:update",
    "vpc:securityGroups:*",
    "vpc:securityGroupRules:*",
    "vpc:floatingIps:*",
    "vpc:publicIps:*",
    "ims:images:create",
    "ims:images:delete",
    "ims:images:get",
    "ims:images:list",
    "ims:images:update",
    "ims:images:upload"
    ],
    "Effect": "Allow"
}
]
}
```

VPC Administrator policy

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "vpc:vpcs:*",
                "vpc:routers:*",
                "vpc:networks:*",
                "vpc:subnets:*",
                "vpc:ports:*",
                "vpc:privateIps:*",
                "vpc:peerings:*",
                "vpc:routes:*",
                "vpc:lbaas:*",
                "vpc:vpns:*",
                "ecs:*:get",
                "ecs:*:list",
                "elb:*:get",
                "elb:*:list"
            ],
            "Effect": "Allow"
        }
    ]
}
```

NAT Admin policy

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "nat:*:*",
                "vpc:*:*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

DEW KeypairFullAccess policy

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "kps:domainKeypairs:*",
                "ecs:serverKeypairs:*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

ELB FullAccess policy

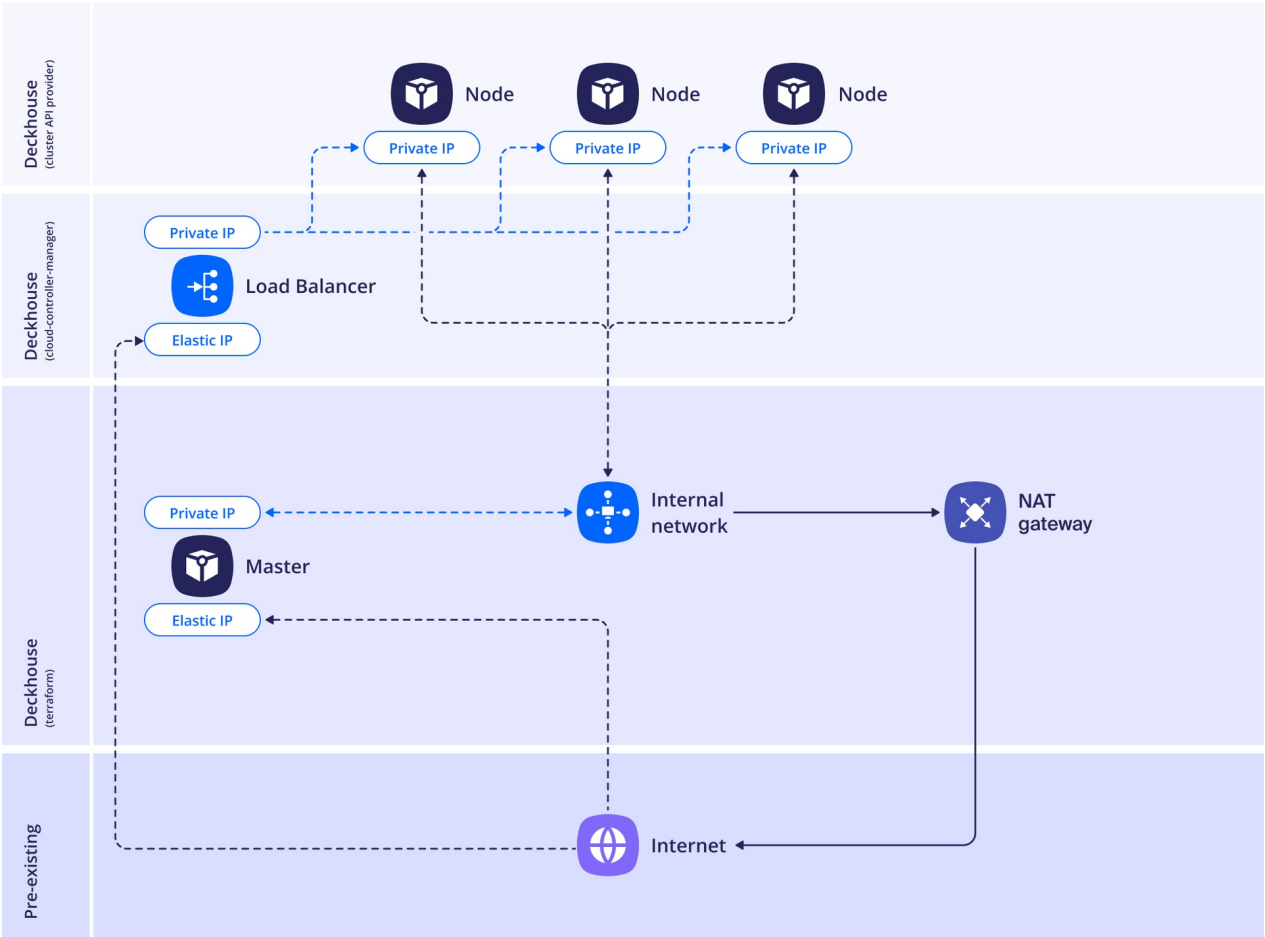
```
{
  "Version": "1.1",
  "Statement": [
      {
          "Action": [
              "elb:*:*",
              "vpc:*:get*",
              "vpc:*:list*",
              "ecs:*:get*",
              "ecs:*:list*"
          ],
          "Effect": "Allow"
      }
  ]
}
```

Layouts and configuration in Huawei Cloud

This section describes the cluster layouts in Huawei Cloud infrastructure and the associated parameters.

Standard

- An internal cluster network is created with a gateway to the public network.
- An Elastic IP can be assigned to the master node.
- Nodes managed by the Cluster API do not have public IP addresses.

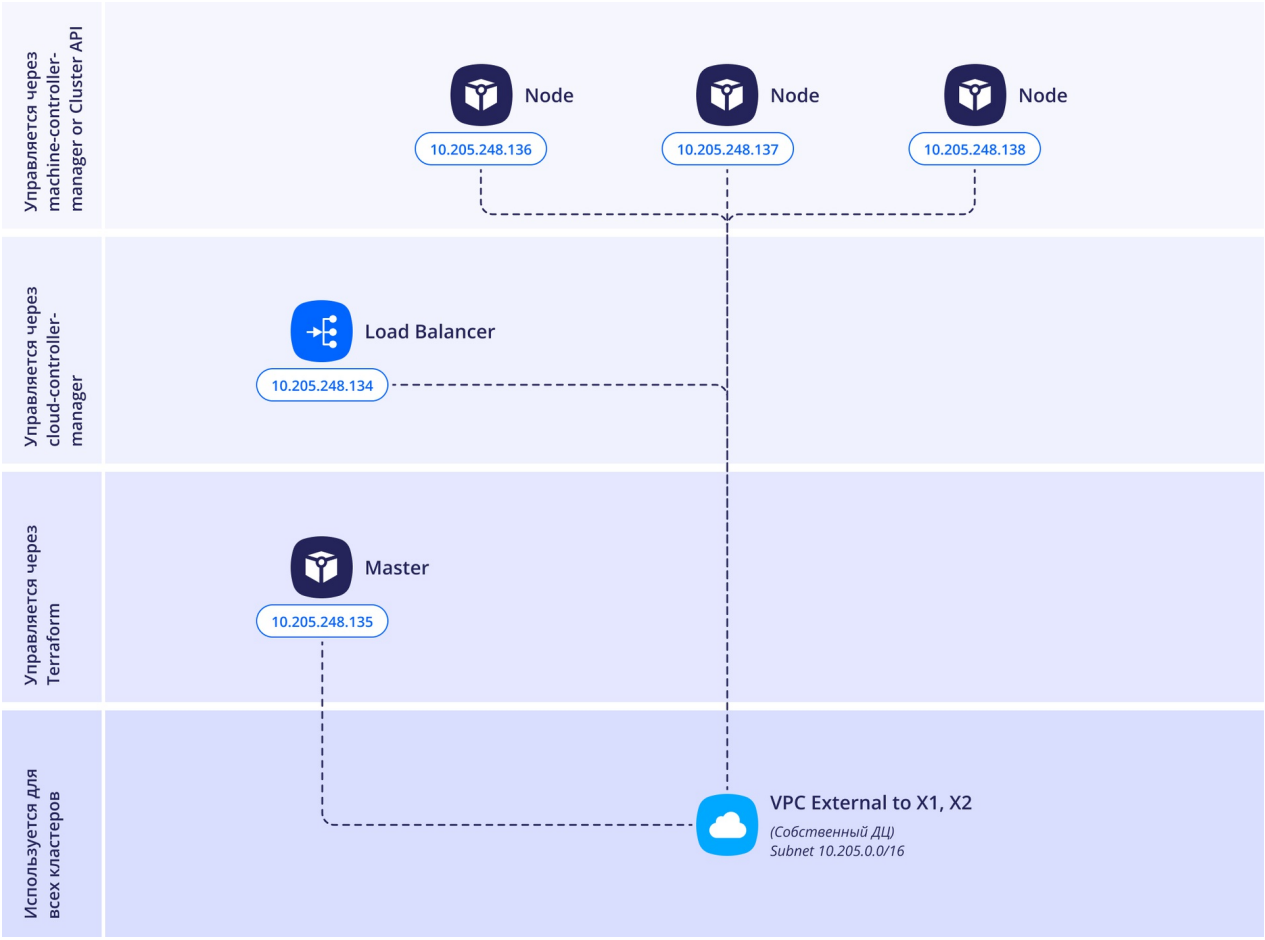


Example configuration:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
standard:
  internalNetworkDNSServers:
    - 8.8.8.8
  internalNetworkCIDR: 192.168.199.0/24
  internalNetworkSecurity: true
  enableEIP: true
provider:
  cloud: hc.sbercloud.ru
  region: ru-moscow-1
  accessKey: "<Access key>"
  secretKey: "<Secret key>"
  projectID: "<Project ID>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    imageName: alt-p11
    flavorName: s7n.xlarge.2
    rootDiskSize: 50
  serverGroup:
    policy: AntiAffinity
  volumeTypeMap:
    ru-moscow-1a: SSD
```

VpcPeering

- An existing VPC network and subnet are used.
- Virtual machines are connected to the specified subnet.



Example configuration:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: VpcPeering
sshPublicKey: "<SSH_PUBLIC_KEY>"
vpcPeering:
  internalNetworkDNSServers:
    - 8.8.8.8
  internalNetworkCIDR: 10.221.128.0/24
  internalNetworkSecurity: true
  subnet: subnet-43b4
```

Storage and load balancing in Huawei Cloud

Storage

Deckhouse Kubernetes Platform provisions disks in Huawei Cloud using the CSI driver. Storage type is configured via parameters in the HuaweiCloudClusterConfiguration resource, specifically the `volumeTypeMap` field.

Example configuration:

```
masterNodeGroup:
  volumeTypeMap:
    ru-moscow-1a: SSD
```

When the `rootDiskSize` parameter is specified, the same disk type is used for the boot (root) volume.

The following parameters can also be set in the HuaweiCloudInstanceClass:

- `rootDiskSize`: Disk size in GiB.
- `rootDiskType`: Type of the root disk (SSD, GPSSD, SAS, etc.).

Example:

```
spec:
  rootDiskSize: 50
  rootDiskType: GPSSD
```

It is recommended that you use the fastest available disk types supported in the cloud provider’s region.

Load balancing

Load balancing in Huawei Cloud is handled via Elastic Load Balancer (ELB) services. To access the load balancer API, the IAM policy must include the `ELB FullAccess` permission.

In the Standard layout, the `enableEIP` parameter is available to assign Elastic IPs to master nodes.

By default, nodes managed by the Cluster API do not receive public IP addresses.

Additional resources

- Module `cloud-provider-huaweicloud` provider configuration
- Module `cloud-provider-huaweicloud` custom resources

Integration with Huawei Cloud services

Deckhouse Kubernetes Platform leverages Huawei Cloud’s capabilities to operate Kubernetes clusters. The following features are supported:

- Resource management in Huawei Cloud via `cloud-controller-manager`.
- Disk provisioning using the CSI driver.
- Integration with the `node-manager` module, allowing the use of HuaweiCloudInstanceClass in NodeGroup.

Working with InstanceClass

The HuaweiCloudInstanceClass resource is used to define the parameters of virtual machines. It is referenced by NodeGroup.

Example resource:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudInstanceClass
metadata:
  name: worker
spec:
  imageName: alt-p11
  flavorName: s7n.xlarge.2
  rootDiskSize: 50
  rootDiskType: SSD
```

Additional resources

- Module `cloud-provider-huaweicloud` custom resources
- Module `node-manager` custom resources

Virtualization systems

Deckhouse Kubernetes Platform (DKP) supports the following virtualization systems:

- Deckhouse Virtualization Platform (DVP);
- VMware Cloud Director;
- VMware vSphere;
- zVirt.

The following sections describe how you can configure integration with each of these virtualization systems.

Integration with Deckhouse Virtualization Platform cloud

Deckhouse Kubernetes Platform integrates with the DVP infrastructure and uses DVPInstanceClass resources to define the characteristics of virtual machines created as part of the cluster.

Key features:

- Management of DVP resources via the `cloud-controller-manager module`
- Provisioning of disks using the CSI storage component
- Integration with the `node-manager` module to support DVPInstanceClass when defining a NodeGroup

> **i** The module is automatically enabled for all cloud clusters deployed in DVP. The module has no configurable settings.

Additional resources

- Module `cloud-provider-dvp` custom resources
- Module `node-manager` documentation

Connection and authorization in Deckhouse Virtualization Platform

To interact with DVP resources, Deckhouse Kubernetes Platform components use the DVP API. To configure access, create a user (ServiceAccount), assign the necessary permissions, and generate a kubeconfig.

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

> ⚠ If the `update-hostname` module of the `cloud-init` package is not disabled, it is recommended to change its run frequency from `always` to `once-per-instance`.
>
> To do this, modify the `update-hostname` module configuration in the `/etc/cloud/cloud.cfg` file:
>
> ```
> cloud_init_modules:
>  ...
>  - [update-hostname, once-per-instance]
>  ...
> ```
>
> The `update-hostname` module can also be disabled completely by removing it from the `cloud_init_modules` module list in the `/etc/cloud/cloud.cfg` file.

Creating a user

Create a new user in the DVP cluster using the following command:

```
d8 k create -f -<<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-demo
  namespace: default
---
apiVersion: v1
kind: Secret
metadata:
  name: sa-demo-token
  namespace: default
  annotations:
    kubernetes.io/service-account.name: sa-demo
type: kubernetes.io/service-account-token
EOF
```

Assigning a role

Assign a role to the created user in the DVP cluster using the following command:

```
d8 k create -f -<<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: sa-demo-rb
  namespace: default
subjects:
  - kind: ServiceAccount
    name: sa-demo
    namespace: default
roleRef:
  kind: ClusterRole
  name: d8:use:role:manager
  apiGroup: rbac.authorization.k8s.io
EOF
```

Generating a kubeconfig

Generate a kubeconfig to be used in the cluster initial configuration file:

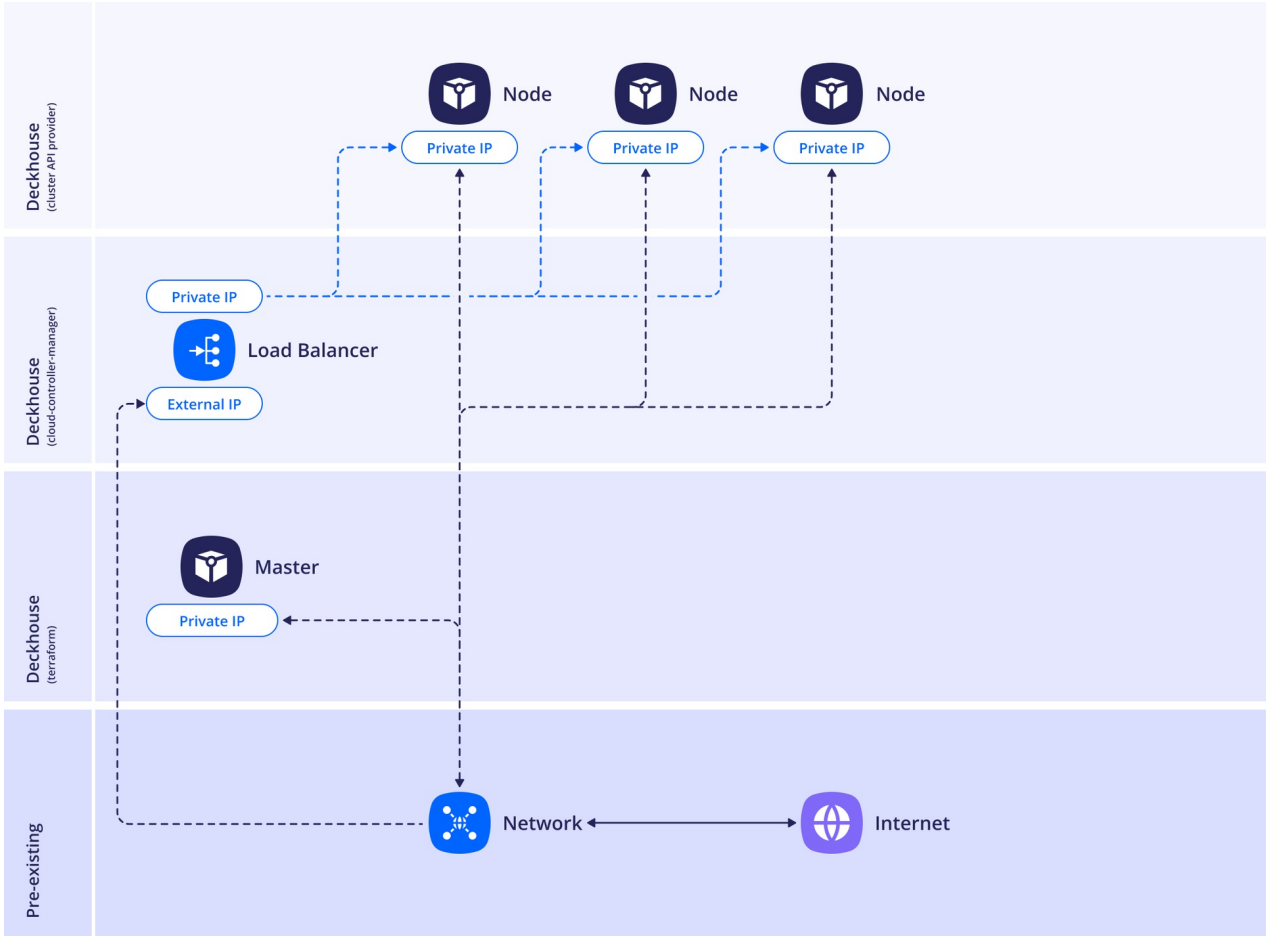
```
cat <<EOF > kubeconfig
apiVersion: v1
clusters:
- cluster:
    server: https://<KUBE-APISERVER-URL>   # Replace this with the actual API server address for the
cluster.
  name: <CLUSTER-NAME>                     # Replace with the cluster name.
contexts:
- context:
    cluster: <CLUSTER-NAME>                # Replace with the cluster name.
    user: sa-demo
    namespace: default
  name: sa-demo-context
current-context: sa-demo-context
kind: Config
preferences: {}
users:
- name: sa-demo
  user:
    token: $(d8 k get secret sa-demo-token -n default -o json | jq -rc .data.token | base64 -d)
EOF
```

Encode the generated kubeconfig file using Base64 encoding (it appears in the initial configuration file as follows):

```
base64 kubeconfig | tr -d '\n'
```

Layouts and configuration in Deckhouse Virtualization Platform

Standard



Example of the layout configuration:

```
---
apiVersion: deckhouse.io/v1
kind: DVPClusterConfiguration
layout: Standard
sshPublicKey: <SSH_PUBLIC_KEY>
masterNodeGroup:
  replicas: 1
  instanceClass:
    virtualMachine:
      cpu:
        cores: 4
        coreFraction: 100%
      memory:
        size: 8Gi
      ipAddresses:
        - Auto
      virtualMachineClassName: generic
    rootDisk:
      size: 50Gi
      storageClass: ceph-pool-r2-csi-rbd-immediate
      image:
        kind: ClusterVirtualImage
        name: ubuntu-2204
    etcdDisk:
      size: 15Gi
      storageClass: ceph-pool-r2-csi-rbd-immediate
provider:
  kubeconfigDataBase64: <KUBE_CONFIG>
  namespace: demo
```

Configuration

Deckhouse uses the DVPClusterConfiguration resource to describe the cluster configuration in DVP.

To modify the configuration in a running cluster, execute:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller edit provider-cluster-configuration
```

After changing node parameters, run the command:

```
dhctl converge
```

Example configuration with multiple parameters:

```
apiVersion: deckhouse.io/v1
kind: DVPClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- zone-a
- zone-b
- zone-c
region: r1
masterNodeGroup:
  replicas: 3
  zones:
  - zone-a
  - zone-b
  - zone-c
  instanceClass:
    virtualMachine:
      cpu:
        cores: 1
        coreFraction: 100%
      memory:
        size: 4Gi
      virtualMachineClassName: generic
      ipAddresses:
      - 10.66.30.100
      - 10.66.30.101
      - 10.66.30.102
      additionalLabels:
        additional-vm-label: label-value
      additionalAnnotations:
        additional-vm-annotation: annotation-value
      tolerations:
      - key: dedicated.deckhouse.io
        operator: Equal
        value: system
      nodeSelector:
        beta.kubernetes.io/os: linux
    rootDisk:
      size: 10Gi
      storageClass: linstor-thin-r1
      image:
        kind: ClusterVirtualImage
        name: ubuntu-2204
    etcdDisk:
      size: 10Gi
      storageClass: linstor-thin-r1
nodeGroups:
- name: worker
  zones:
  - zone-a
  - zone-b
  - zone-c
  replicas: 1
  instanceClass:
    virtualMachine:
      cpu:
        cores: 4
        coreFraction: 100%
      memory:
        size: 8Gi
      virtualMachineClassName: generic
    rootDisk:
      size: 10Gi
      image:
        kind: ClusterVirtualImage
        name: ubuntu-2204
provider:
  kubeconfigDataBase64: ZXhhbXBsZQo=
  namespace: default
```

Additional resources

- Module `cloud-provider-dvp` provider configuration

Connection and authorization in VMware Cloud Director

Resource preparation

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

To manage resources in VCD using the “Deckhouse Kubernetes Platform”, the following resources must be configured in the system:

- Organization
- VirtualDataCenter
- vApp (for the “Standard” placement scheme)
- StoragePolicy
- SizingPolicy
- Network (for the “Standard” placement scheme)
- EdgeRouter
- Catalog

The Organization, VirtualDataCenter, StoragePolicy, SizingPolicy, EdgeRouter, and Catalog resources must be provided by your VMware Cloud Director service provider.

The Network (internal network) can be configured by your VMware Cloud Director service provider or by yourself. When using the “StandardWithNetwork” placement scheme, the network is created automatically. Below is a method for manually setting up an internal network.

User permissions

For access to the VMware Cloud Director API, one of the following options is sufficient:

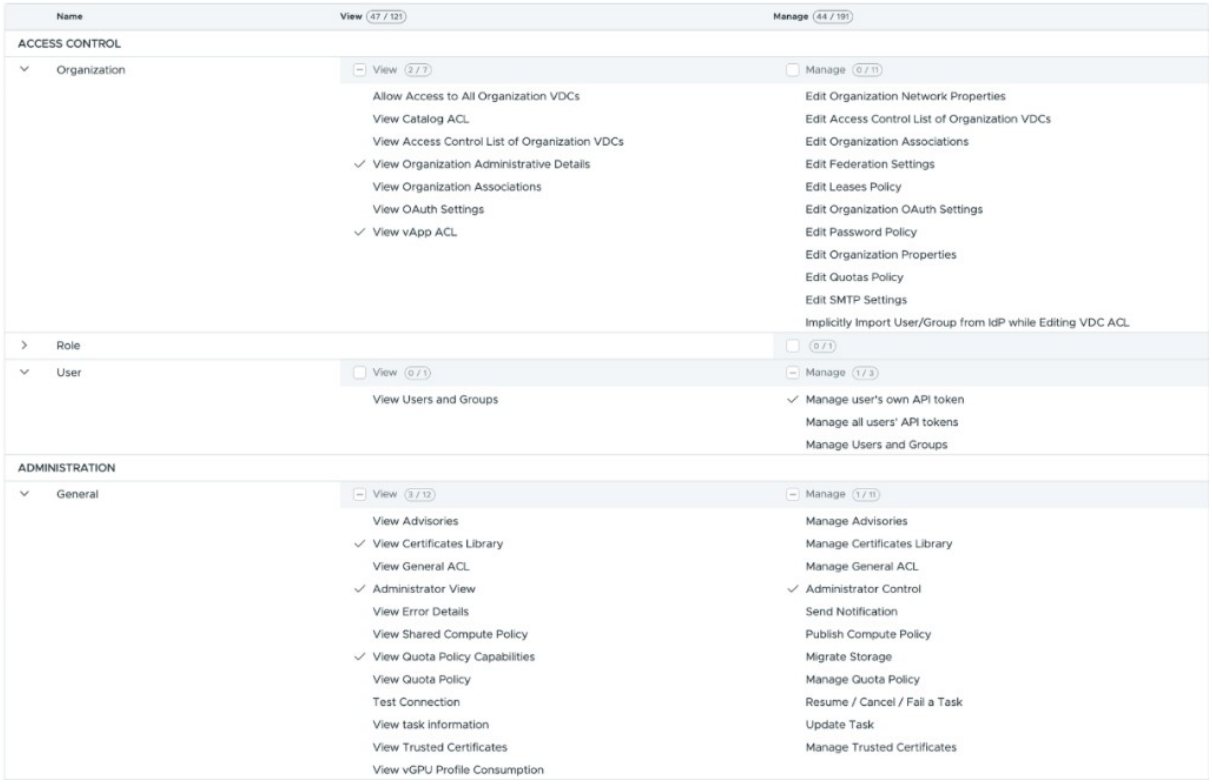
- The `Organization Administrator` role with the `Preserve All ExtraConfig Elements During OVF Import and Export` right. This option does not require configuring a separate role, but grants the user full administrative access to the organization.
- A custom role with the set of rights listed below. This set is sufficient to create a cluster, provision nodes (including CloudEphemeral with static addressing and DHCP), and work with CSI disks and StorageClass without granting additional administrative rights.

In VMware Cloud Director, user permissions come from the role and from a **Rights Bundle** — a set of rights published to the organization. Add the `Preserve All ExtraConfig Elements During OVF Import and Export` right both to the role and to the organization’s Rights Bundle.

⚠ Without the `Preserve All ExtraConfig Elements During OVF Import and Export` right in both the role and the organization’s Rights Bundle, `userdata` is not passed to the virtual machines of ephemeral nodes.

To create a custom role, enable the rights as shown below. Right names match the VCD UI.

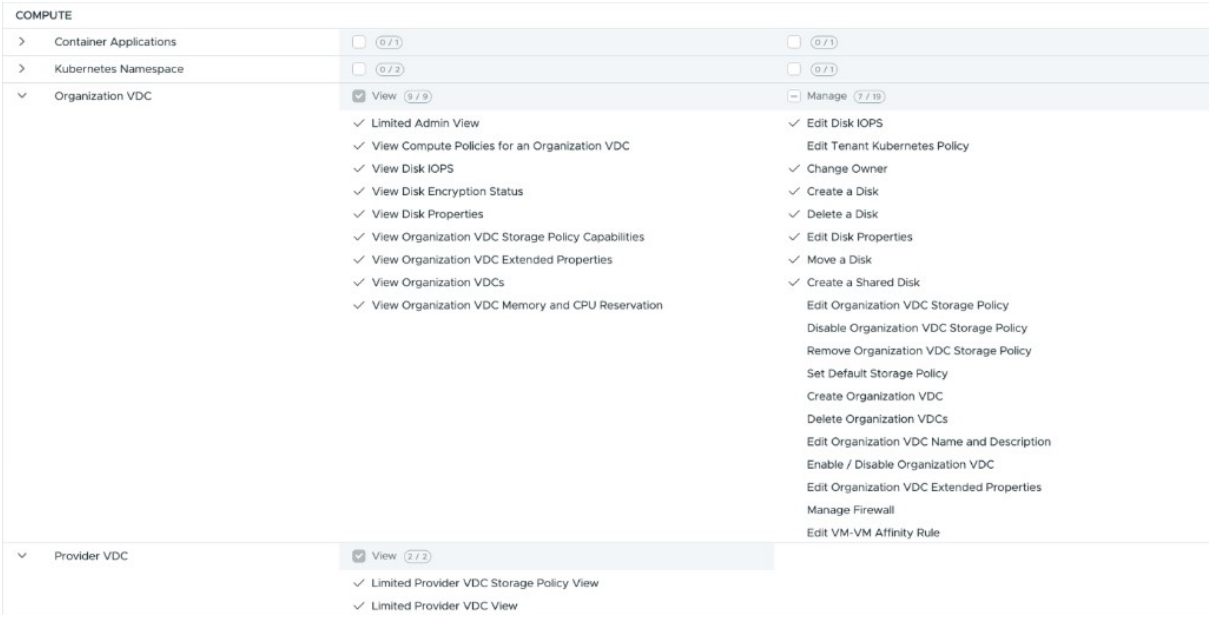
1. In the “ACCESS CONTROL” and “ADMINISTRATION” sections, enable the following rights:
   - Organization → View: `View Organization Administrative Details`, `View vApp ACL`
   - User → Manage: `Manage user's own API token`
   - General → View: `View Certificates Library`, `Administrator View`, `View Quota Policy Capabilities`
   - General → Manage: `Administrator Control`



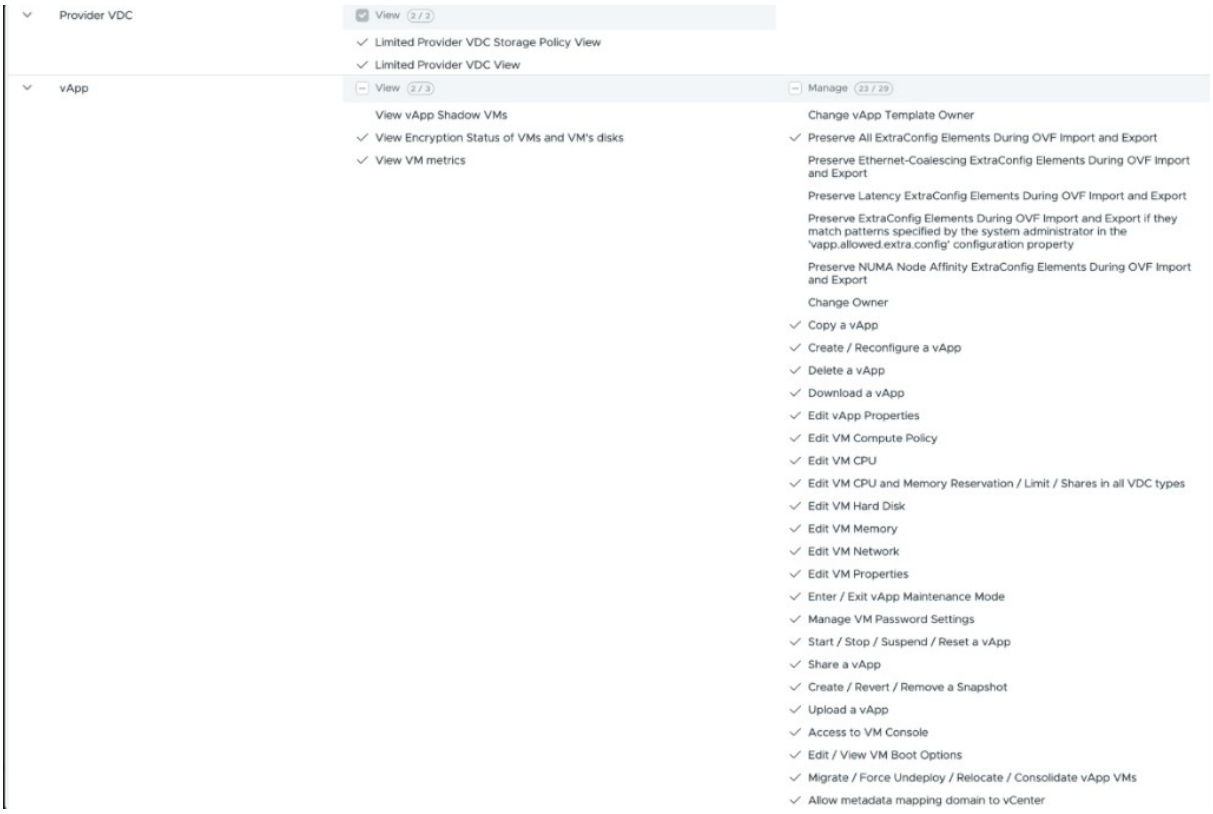
2. In the “COMPUTE” → “Organization VDC” section, enable all View rights and the following Manage rights:

   - `Edit Disk IOPS`
   - `Edit Tenant Kubernetes Policy`
   - `Change Owner`
   - `Create a Disk`
   - `Delete a Disk`
   - `Edit Disk Properties`
   - `Move a Disk`
   - `Create a Shared Disk`

   In the “Provider VDC” → View section, enable `Limited Provider VDC Storage Policy View` and `Limited Provider VDC View`:



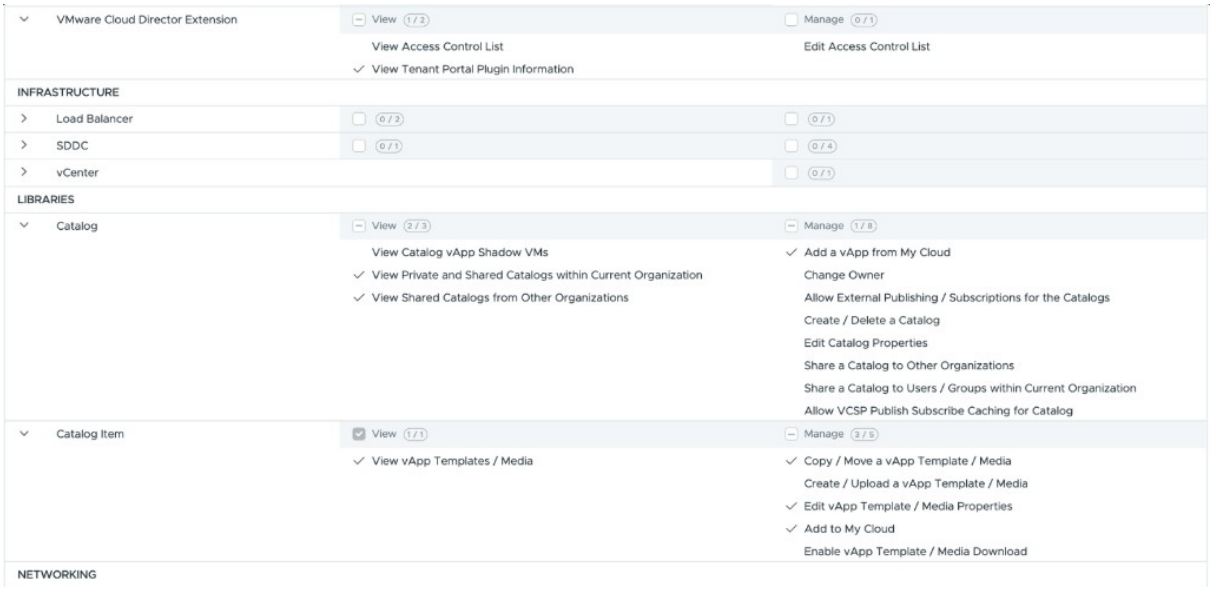
3. In the “COMPUTE” → “vApp” section, enable the rights as shown in the screenshot. Make sure that `Preserve All ExtraConfig Elements During OVF Import and Export` is enabled. In the verified role, View includes `View Encryption Status of VMs and VM's disks` and `View VM metrics`; `View vApp Shadow VMs` and the other Preserve ExtraConfig options are not enabled:



4. In the “LIBRARIES” section, enable the following rights:

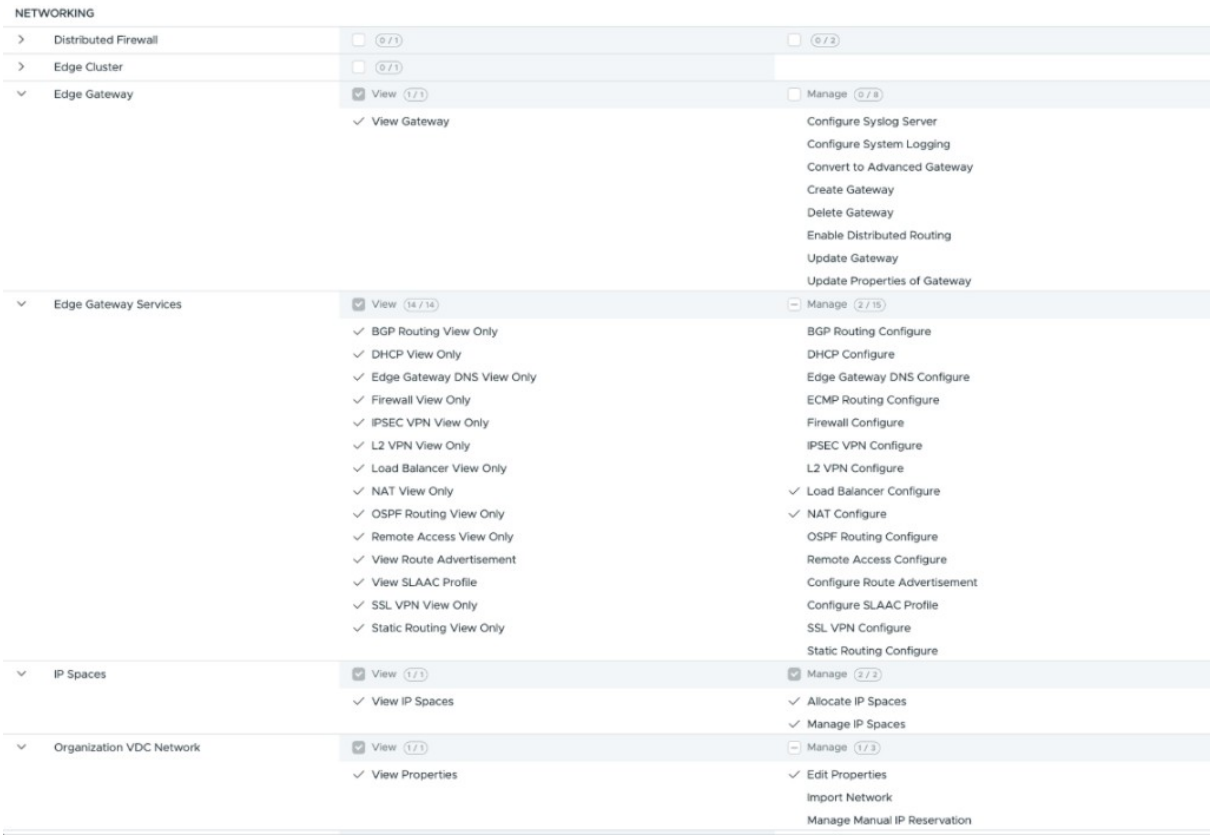
   - Catalog → View: `View Private and Shared Catalogs within Current Organization`, `View Shared Catalogs from Other Organizations` (to use images from shared catalogs)
   - Catalog → Manage: `Add a vApp from My Cloud`
   - Catalog Item → View: `View vApp Templates / Media`
   - Catalog Item → Manage: `Copy / Move a vApp Template / Media`, `Edit vApp Template / Media Properties`, `Add to My Cloud`

   In the “VMware Cloud Director Extension” → View section, enable `View Tenant Portal Plugin Information`:

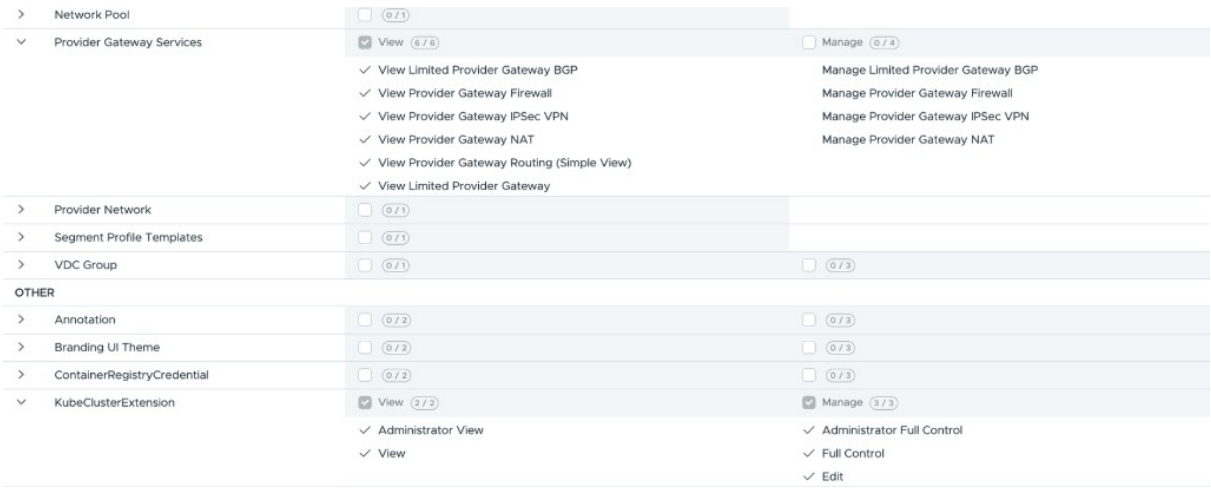


5. In the “NETWORKING” section, enable the following rights:

   - Edge Gateway → View: `View Gateway`
   - Edge Gateway Services → View: all items
   - Edge Gateway Services → Manage: `Load Balancer Configure`, `NAT Configure`
   - IP Spaces → View: `View IP Spaces`
   - IP Spaces → Manage: `Allocate IP Spaces`, `Manage IP Spaces`
   - Organization VDC Network → View: `View Properties`
   - Organization VDC Network → Manage: `Edit Properties` (required to create networks in the organization)



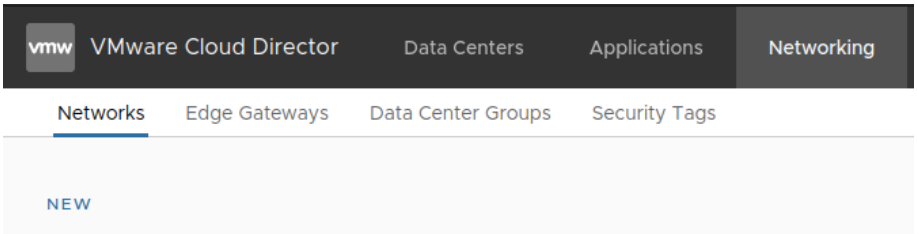
6. In the “Provider Gateway Services” section, enable all View rights. In the “KubeClusterExtension” section, enable all View and Manage rights:



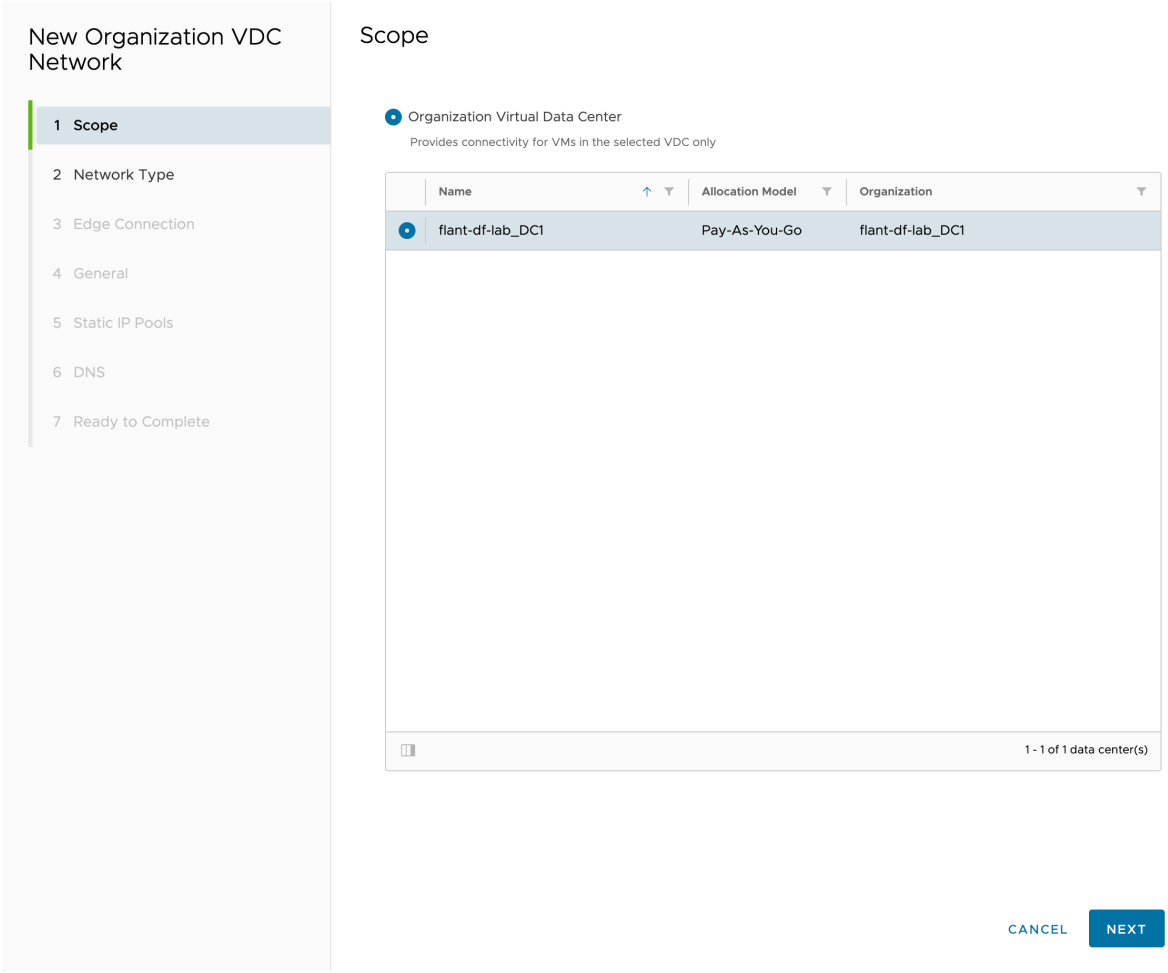
Adding a network

> **i** This instruction applies only to the “Standard” placement scheme.

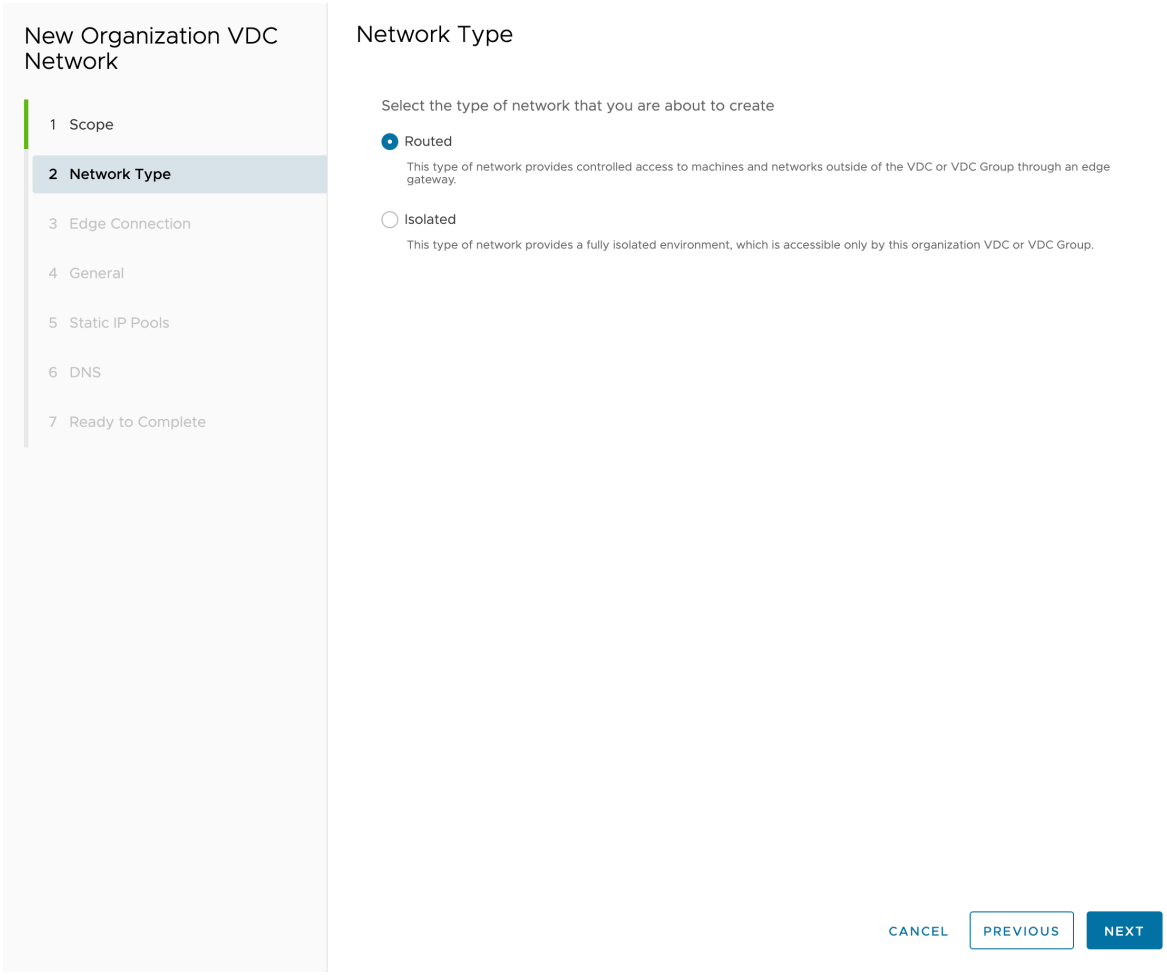
1. Go to the “Networking” tab and click “NEW”:



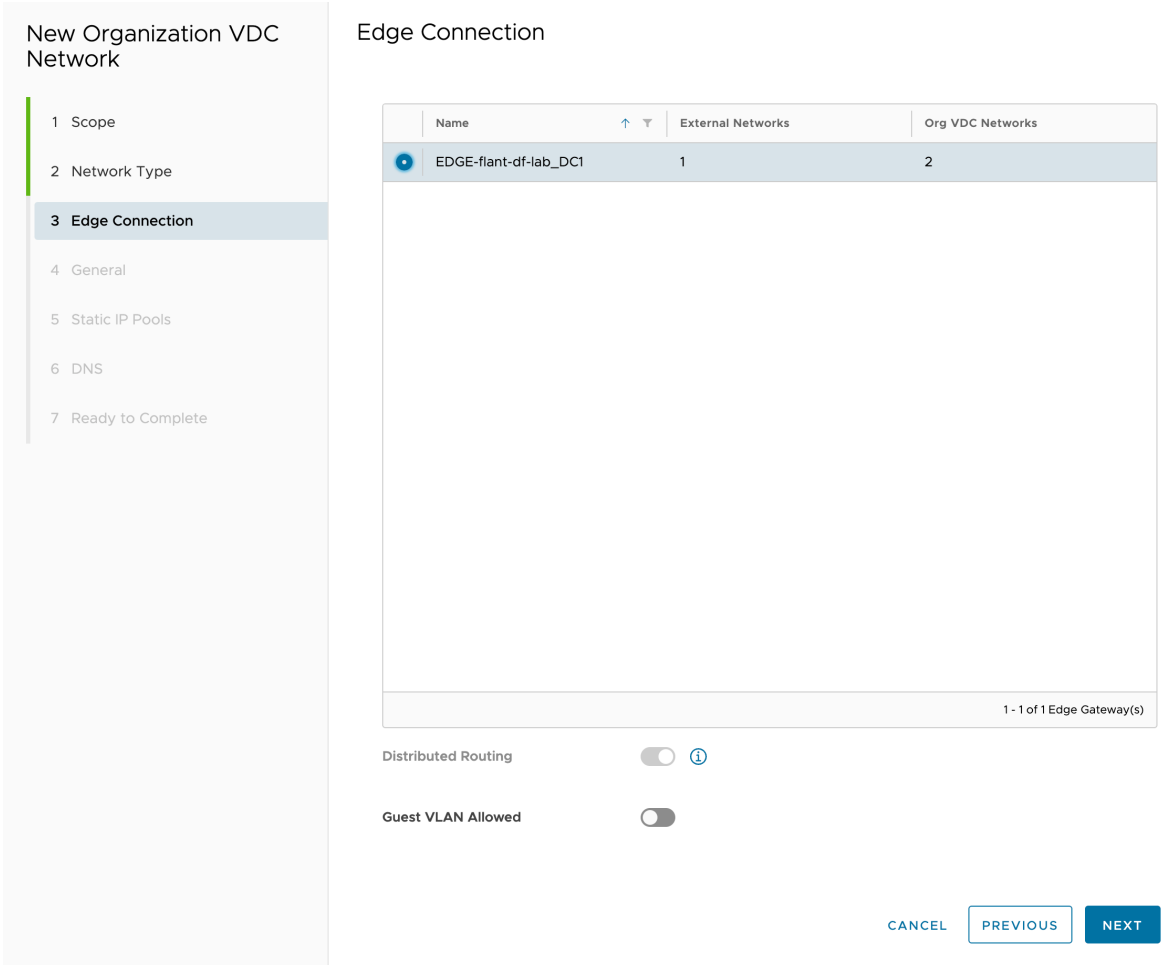
2. Select the desired “Data Center”:



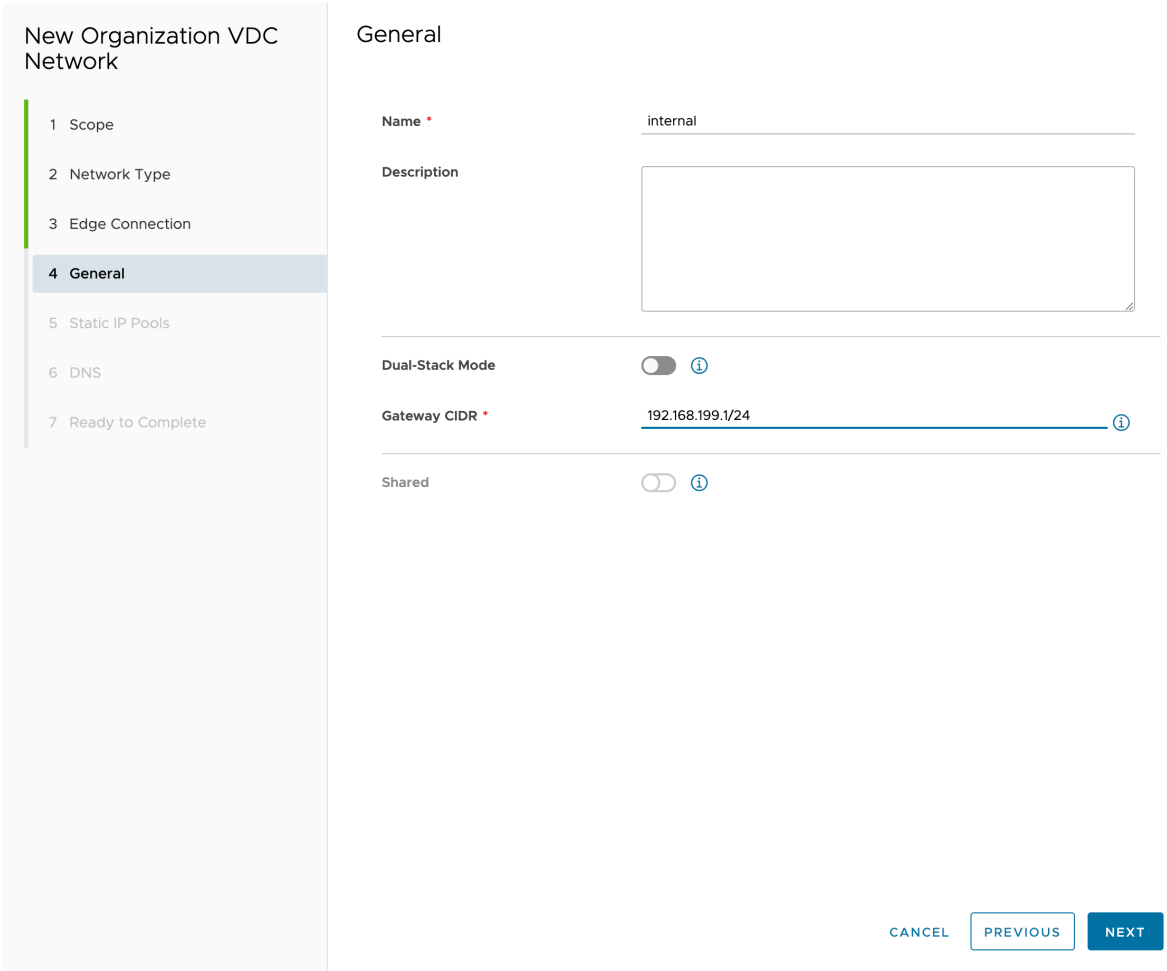
3. In the “Network type” step, select “Routed”:



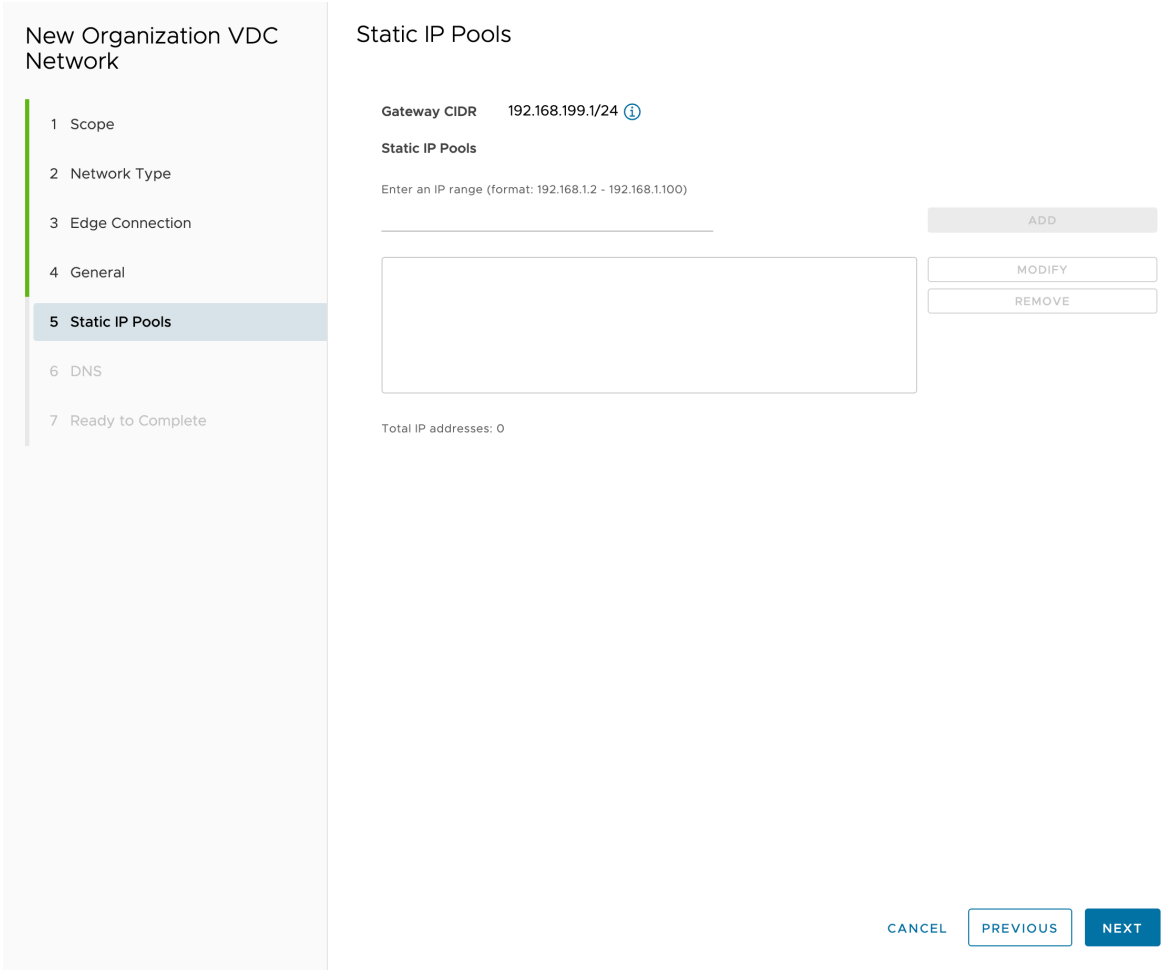
4. Connect the “EdgeRouter” to the network:



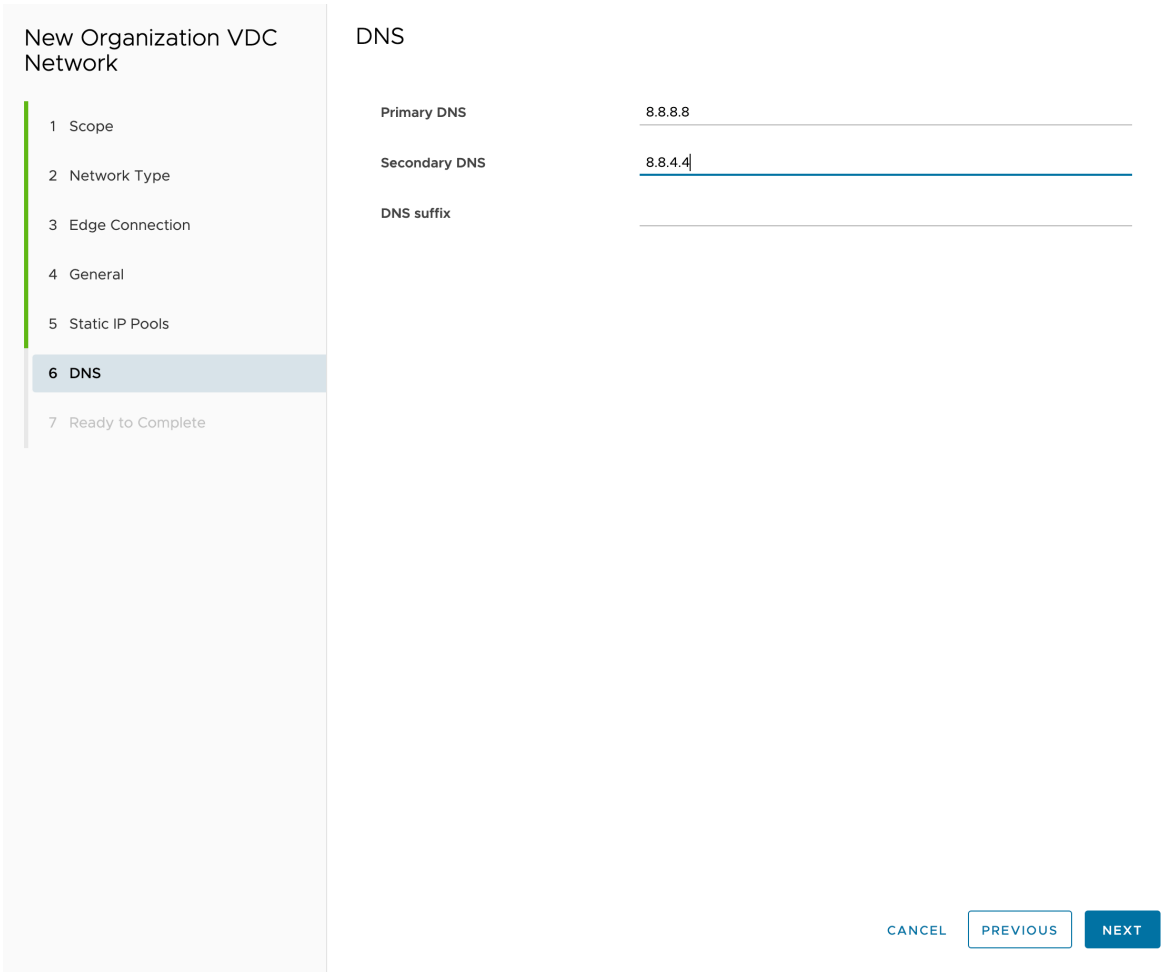
5. Specify the network name and CIDR:



6. Do not add “Static IP Pools” since DHCP will be used:



7. Specify DNS server addresses:



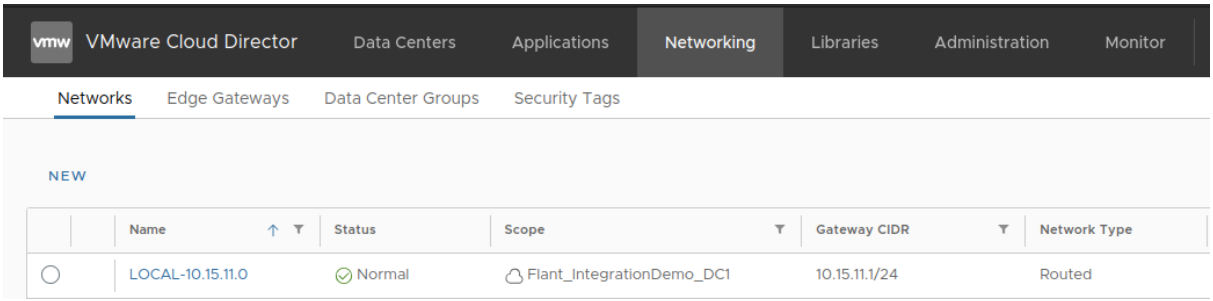
DHCP setup

> **i** This instruction applies only to the “Standard” placement scheme.

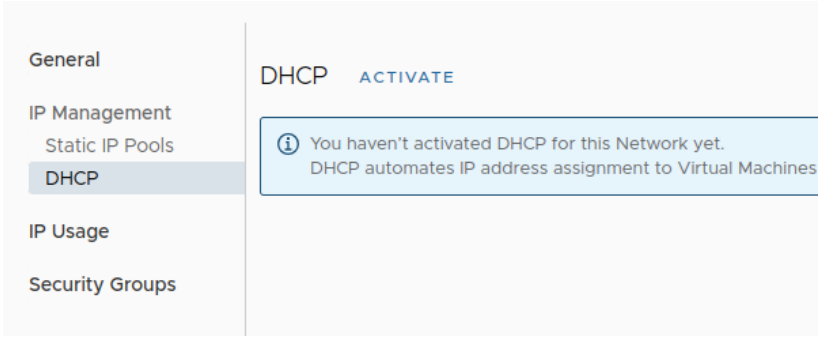
To dynamically provision nodes, enable the DHCP server for the internal network.

> **i** We recommend reserving the beginning of the address range for system workloads (control plane, frontend nodes, system nodes), and using the rest for the DHCP pool.
> For example, for a “/24” network, reserving 20 addresses for system workloads is sufficient.

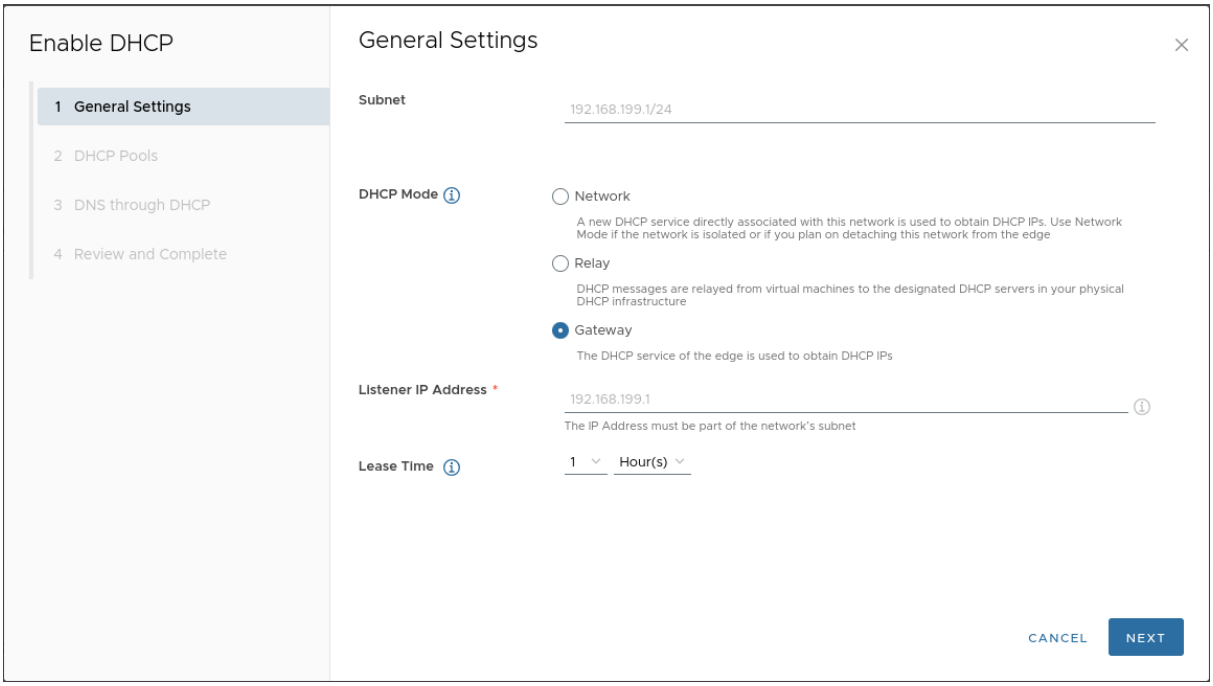
1. Go to the “Networking” tab and open the created network:



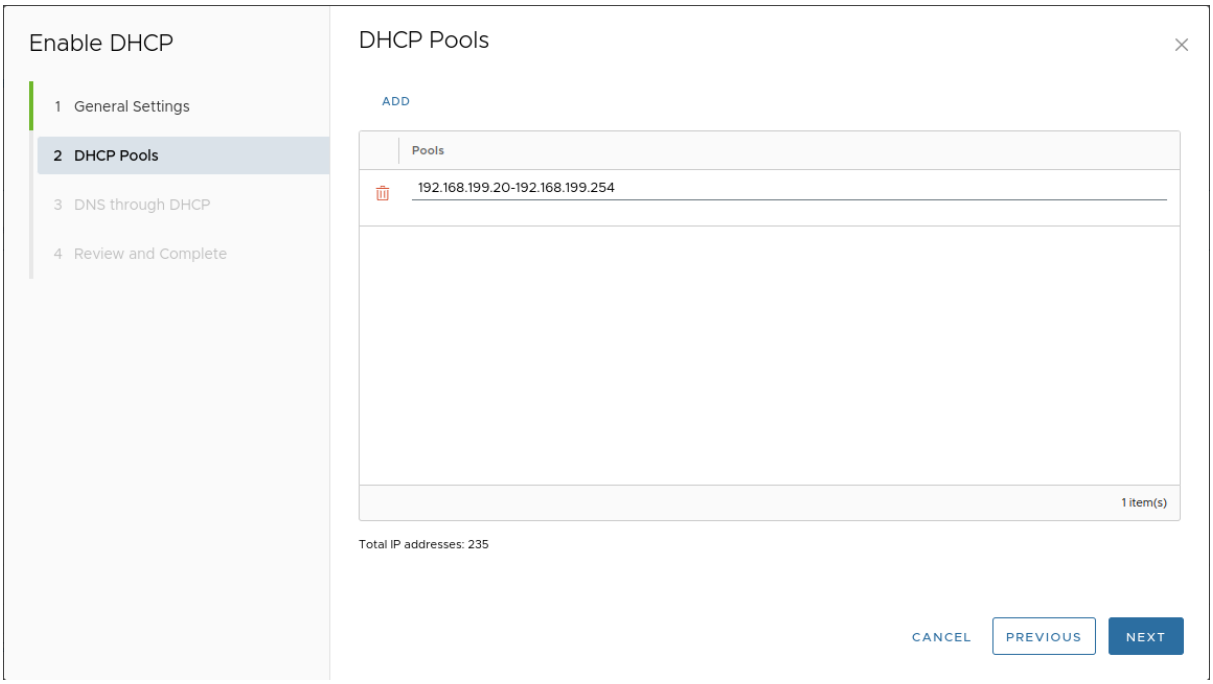
2. In the opened window, select “IP Management” → “DHCP” → “Activate”:



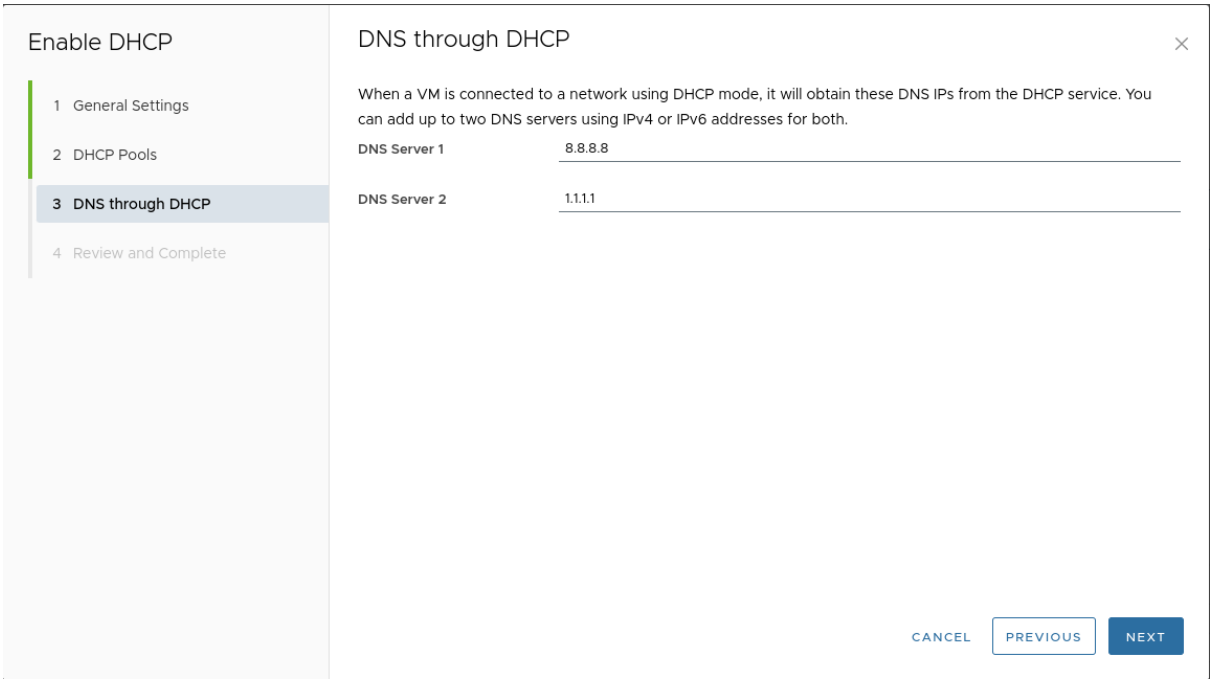
3. In the “General settings” tab, configure parameters as shown in the example:



4. Add a pool:



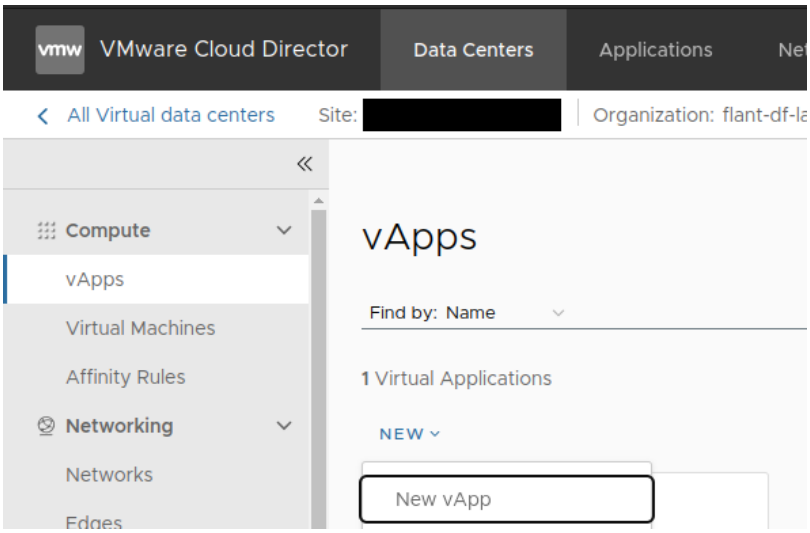
5. Specify DNS server addresses:



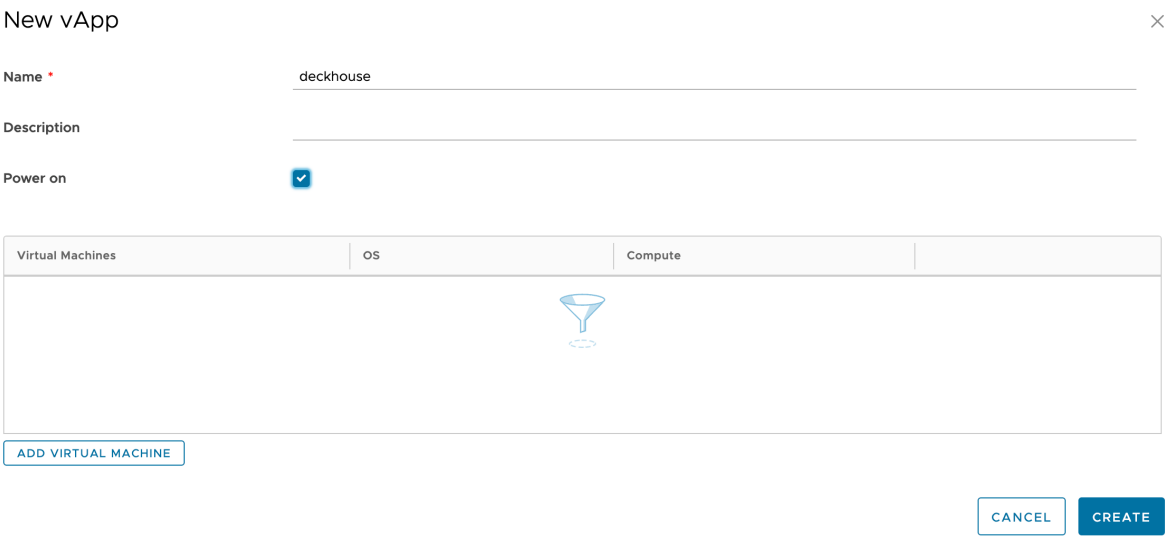
Adding a vApp

> **i** This instruction applies only to the “Standard” placement scheme.

1. Go to “Data Centers” → “vApps” → “NEW” → “New vApp”:



2. Specify a name and enable the vApp:

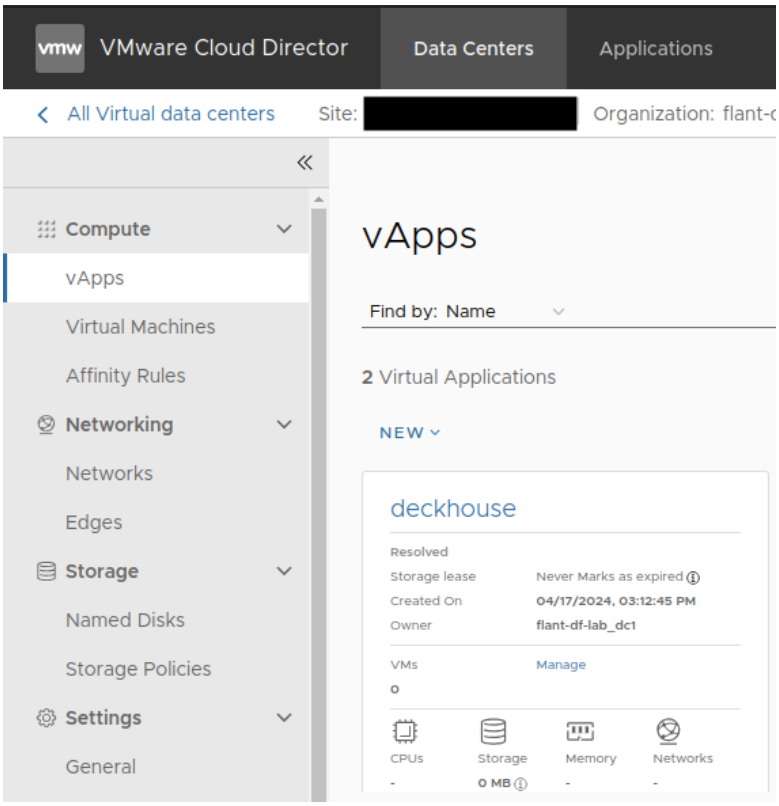


Adding a network to a vApp

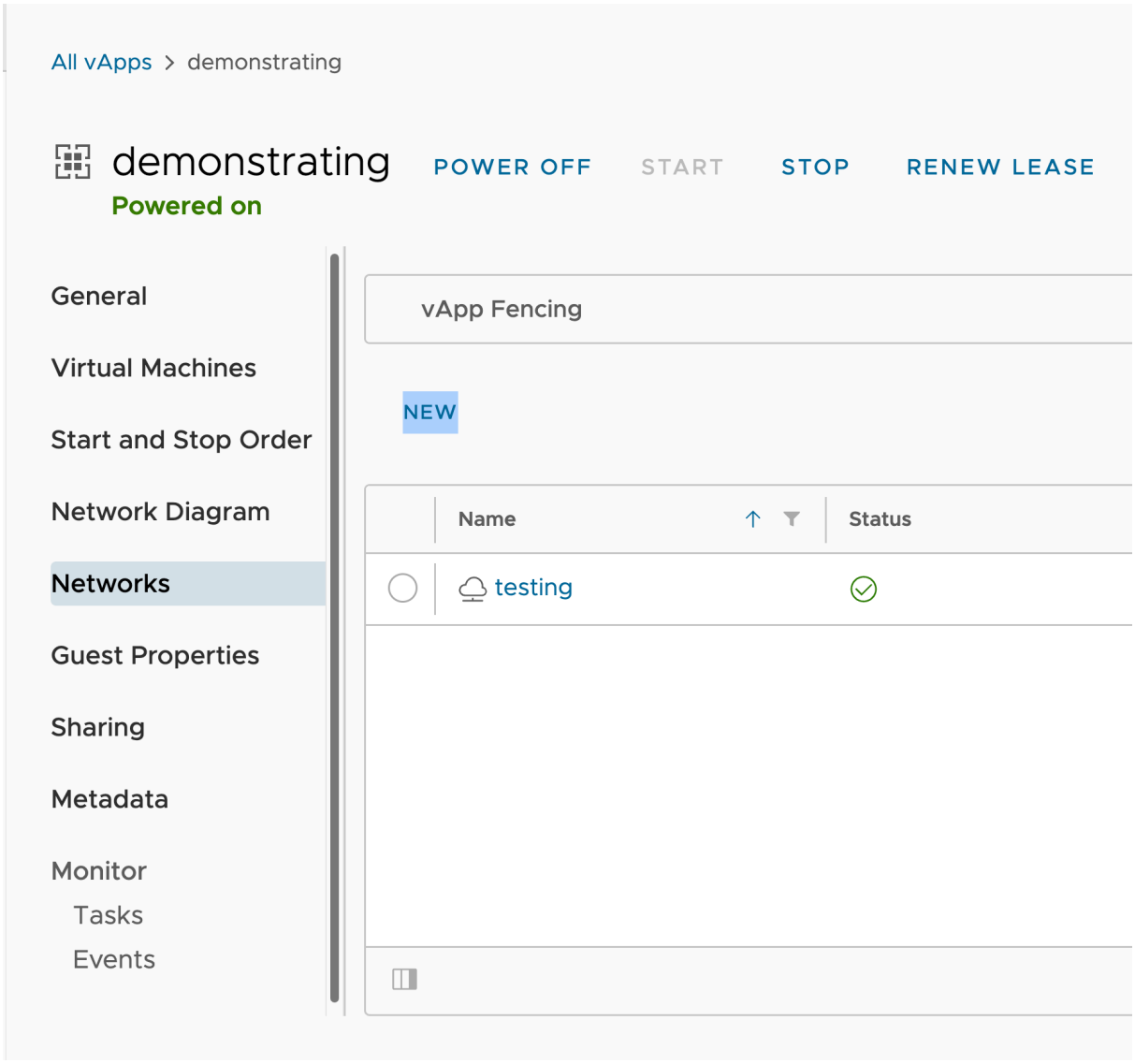
> **i** This instruction applies only to the “Standard” placement scheme.

After creating the vApp, attach the created internal network to it.

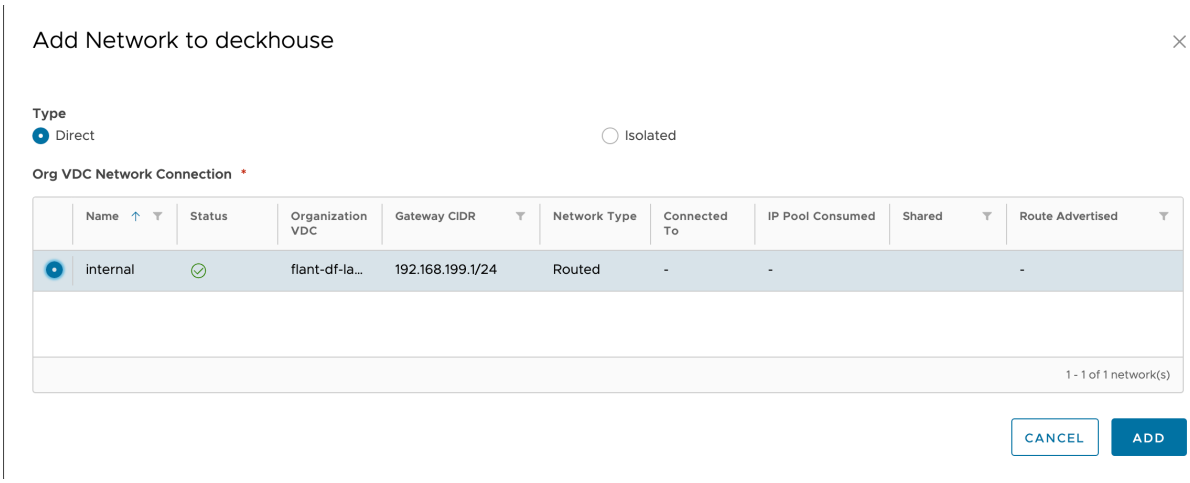
1. Go to “Data Centers” → “vApps” and open the desired vApp:



2. Go to the “Networks” tab and click “NEW”:



3. In the pop-up window, select “Direct” type and choose the network:

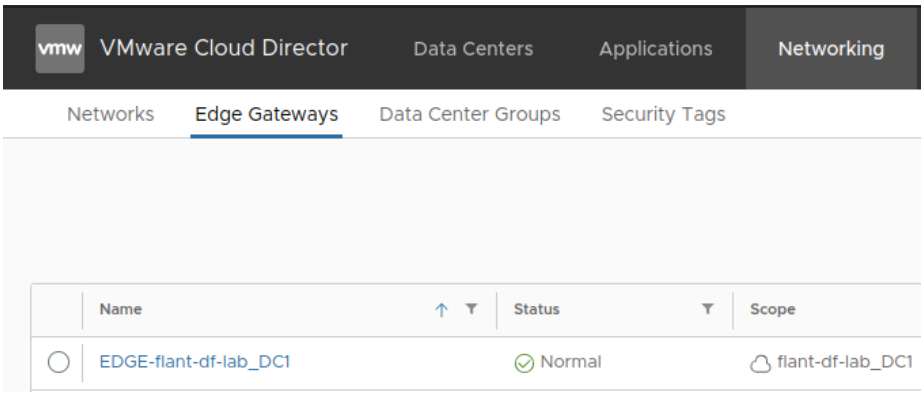


Incoming traffic

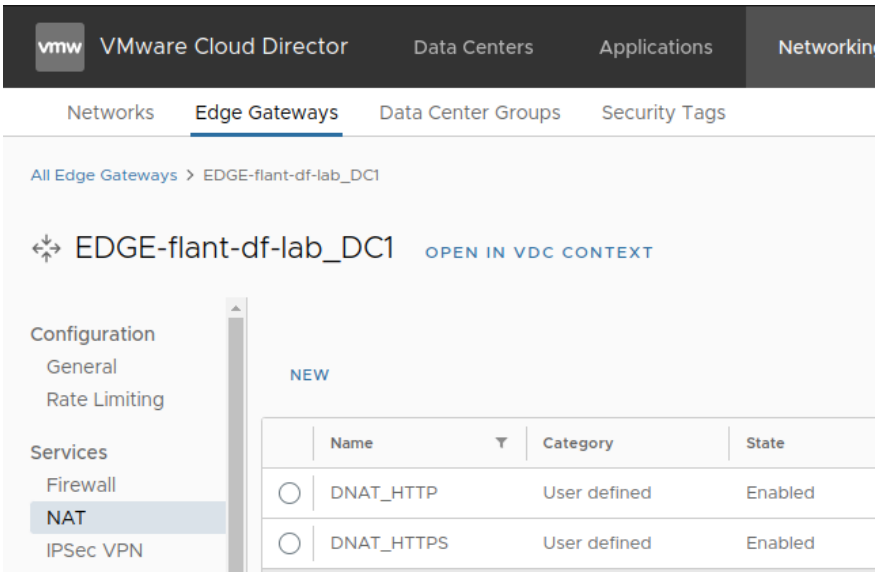
Incoming traffic must be directed to the edge router (ports “80”, “443”) using DNAT rules to the allocated address in the internal network.
This address is managed by MetalLB in L2 mode on dedicated frontend nodes.

Configuring DNAT/SNAT rules on the Edge Gateway

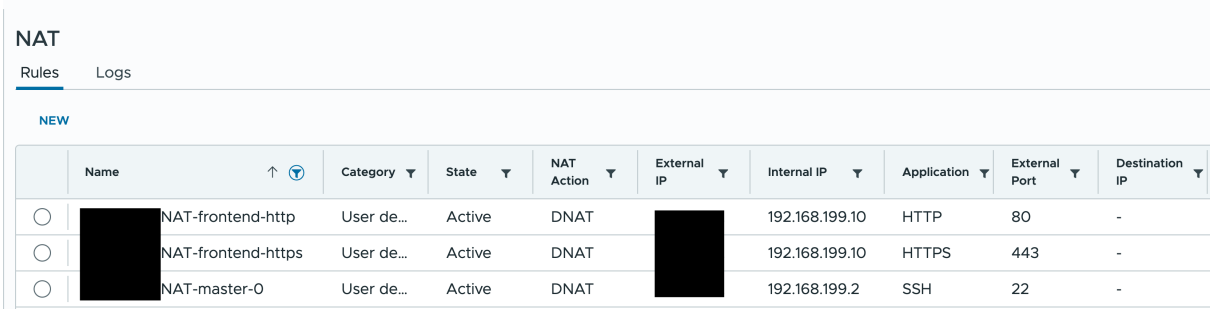
1. Go to “Networking” → “Edge Gateways”, open the edge gateway:



2. Go to “Services” → “NAT”:



3. Add the following rules:



The first two rules are for incoming traffic, and the third is for SSH access to the control plane node (required for installation).

4. To allow virtual machines to access the internet, configure SNAT rules as in the example:



This rule allows VMs from the “192.168.199.0/24” subnet to access the internet.

Let’s Encrypt certificate issuance

> **i** This instruction is applicable only for the `Standard` deployment layout.

When using `cert-manager` with the HTTP-01 ACME challenge type, you may need to configure Hairpin NAT (NAT loopback) on the Edge Gateway.

This is required when the Ingress domain name resolves to the external IP address of the Edge Gateway, while requests to that address are made from **inside** the cluster (from the node network). In this case, the traffic must be correctly routed back to the internal network to the Ingress address (for example, the MetalLB address).

This is important for issuing Let’s Encrypt certificates because cert-manager performs a preliminary self-check of the challenge URL before sending a request to the ACME provider. If the URL is not reachable from inside the cluster, certificate issuance will not start, even if the address is accessible from outside.

Example (cluster network `192.168.199.0/24`):

- internal node network: `192.168.199.0/24`
- external Edge Gateway IP address: `194.117.83.19`
- internal Ingress address (for example, MetalLB): `192.168.199.251`.

In this case, you must configure Hairpin NAT for traffic from the `192.168.199.0/24` network that is sent to the external IP `194.117.83.19`, with translation to the internal Ingress address `192.168.199.251`. Apply Hairpin NAT rules (DNAT and SNAT if needed) to the **local interface** of the Edge Gateway, not to the Uplink.

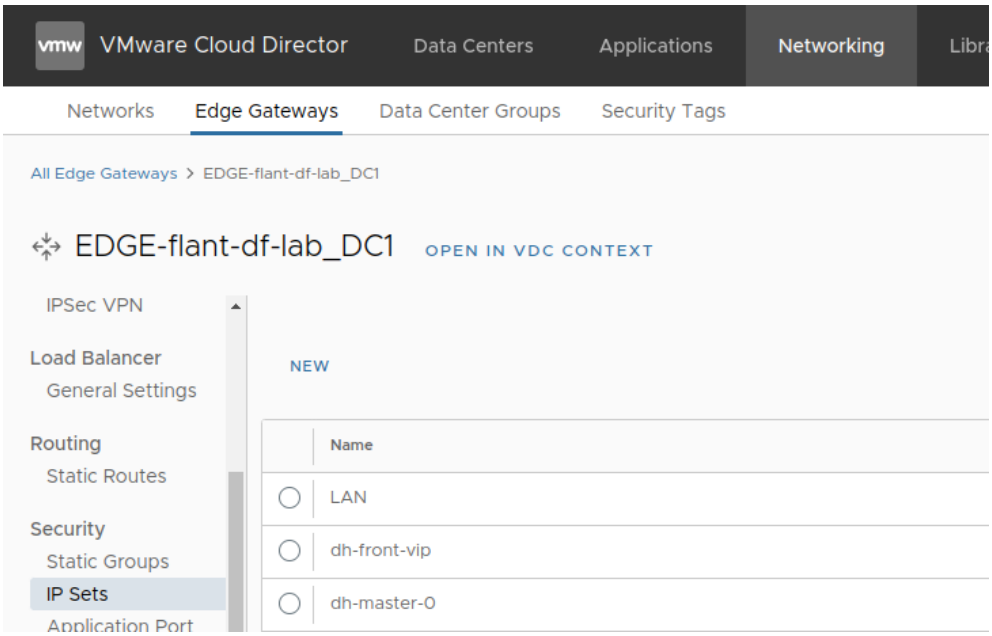
> **i** Hairpin NAT configuration may be required for Edge Gateway based on NSX-V.
>
> For NSX-T, the loopback scenario is often supported by default, but the actual behavior depends on the VMware Cloud Director configuration.

Firewall setup

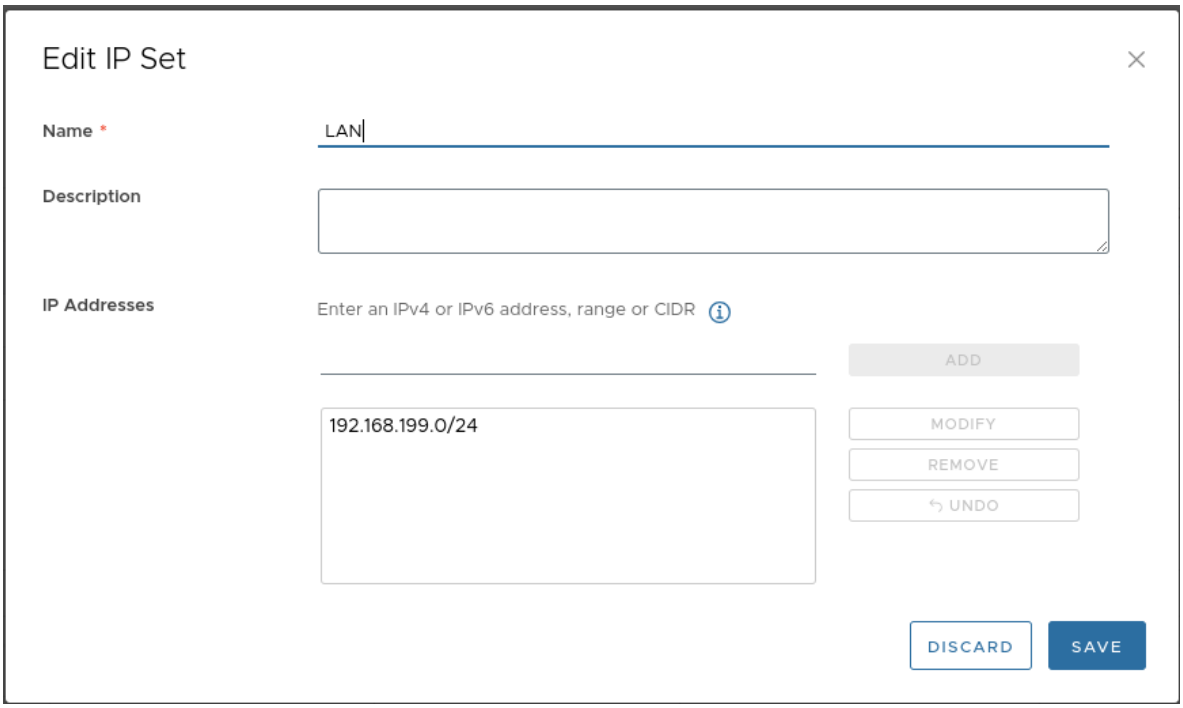
> **i** This instruction applies only to the “Standard” placement scheme.

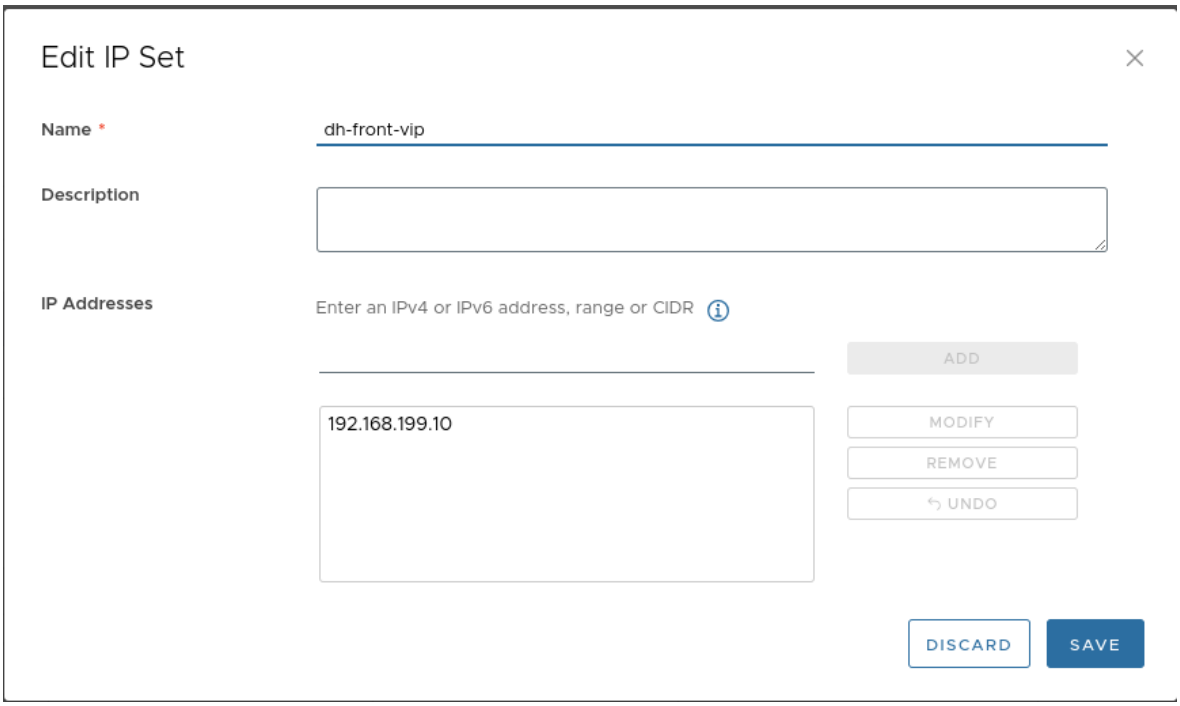
After configuring DNAT, configure the firewall. Start by setting up IP sets.

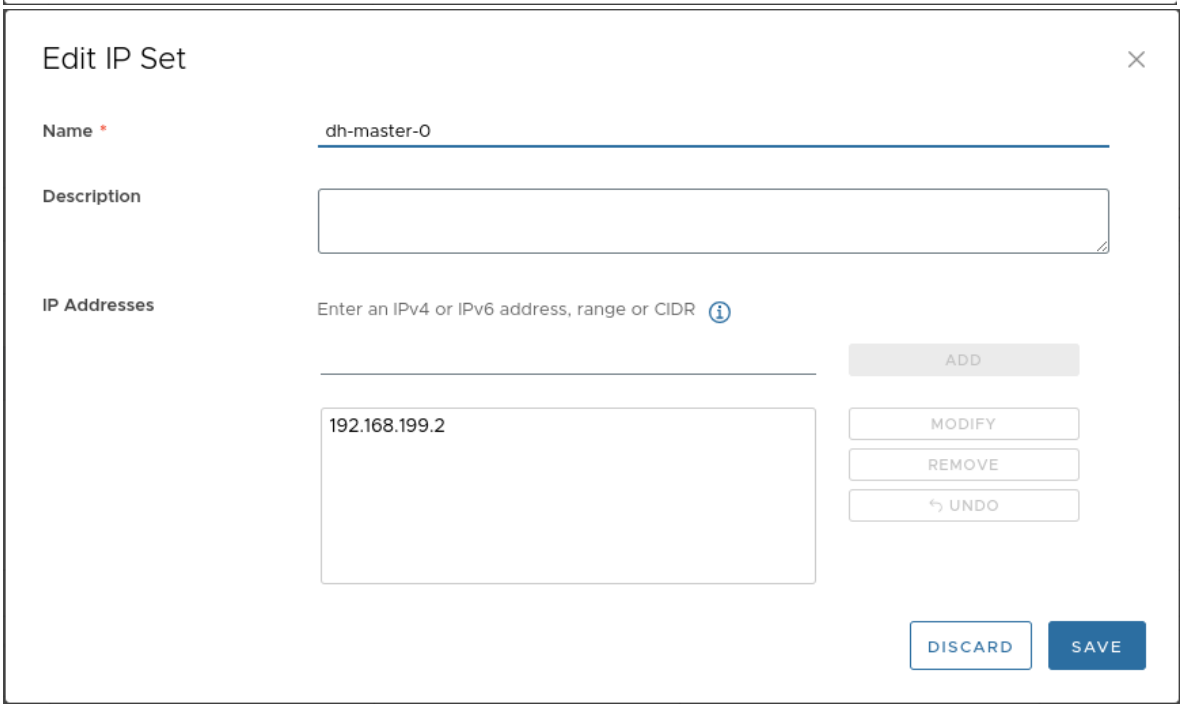
1. Go to “Security” → “IP Sets”:



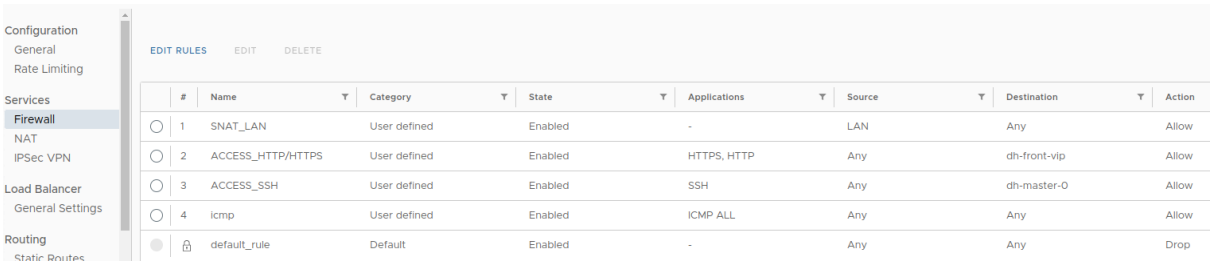
2. Create the following IP set (assuming the MetalLB address will be “.10” and the control plane node “.2”):







3. Add the following firewall rules:



Virtual machine template

⚠ The provider has been tested only with virtual machine templates based on “Ubuntu 22.04”.

⚠ Disable vApp/Guest OS Customization (Guest Customization, vApp Customization, and similar mechanisms) for the cluster template and virtual machines in VMware Cloud Director. DKP performs the initial node provisioning via `cloud-init` (OVF/VMware GuestInfo datasource). If customization is enabled, it may conflict with `cloud-init` and result in incorrect node initialization.

⚠ To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:

- Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- `CgroupsV2` support.
- Systemd version `244` or newer.
- `erofs` kernel module support.

For more information, see the documentation.

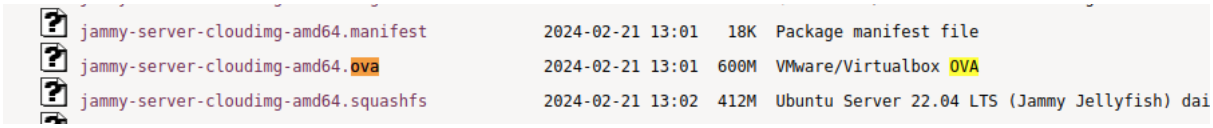
⚠ The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

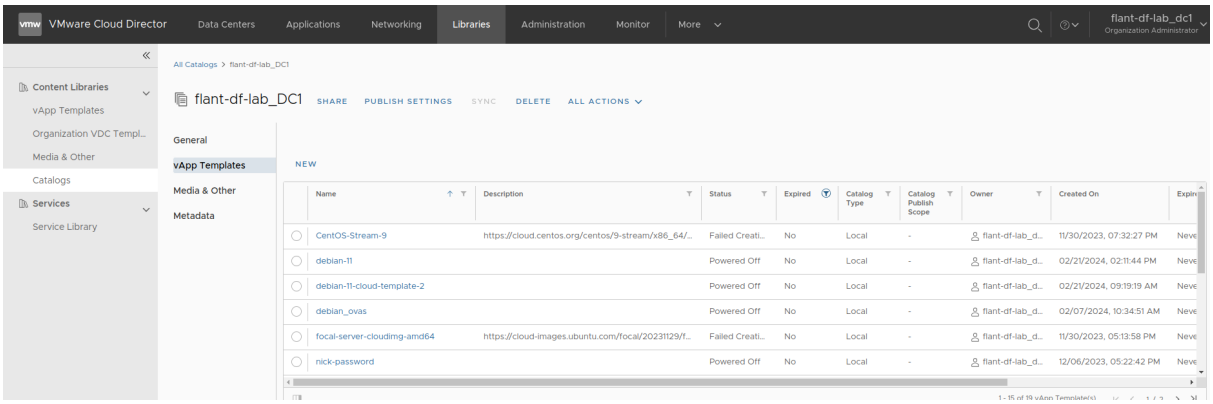
In the example, an “OVA” file provided by Ubuntu is used, with two modifications.
These modifications are required for proper provisioning of CloudPermanent nodes and to allow attaching disks created by CSI.

Preparing the template from an OVA file

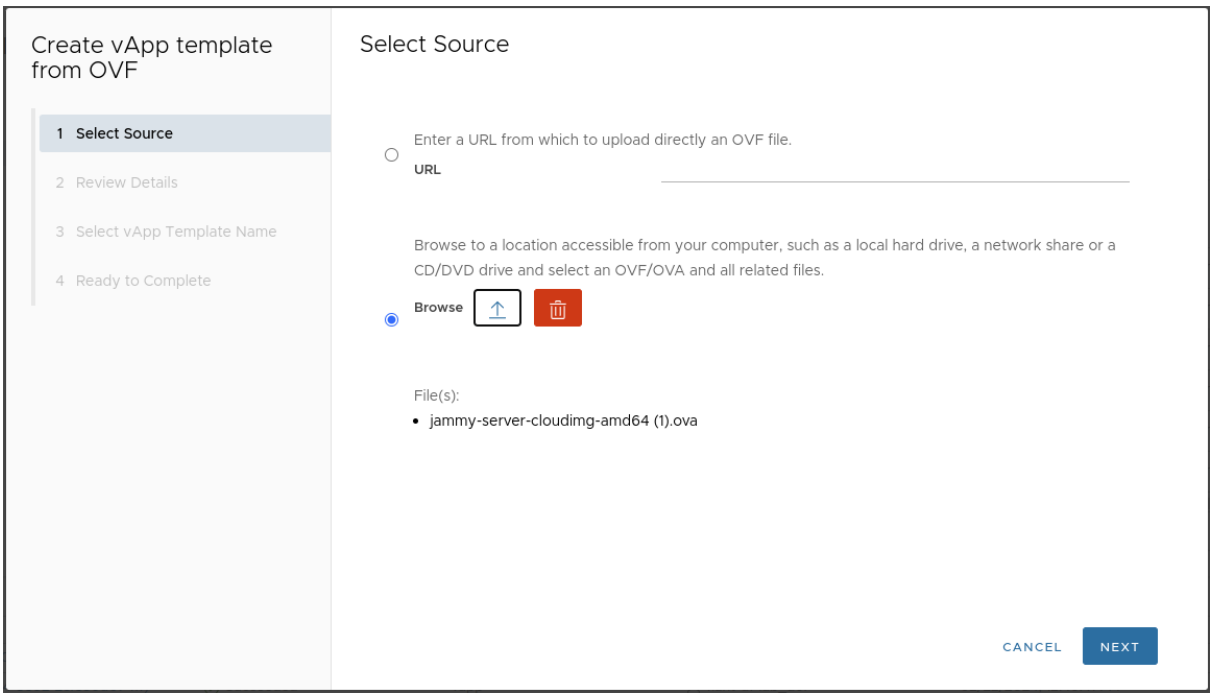
1. Download the OVA file:

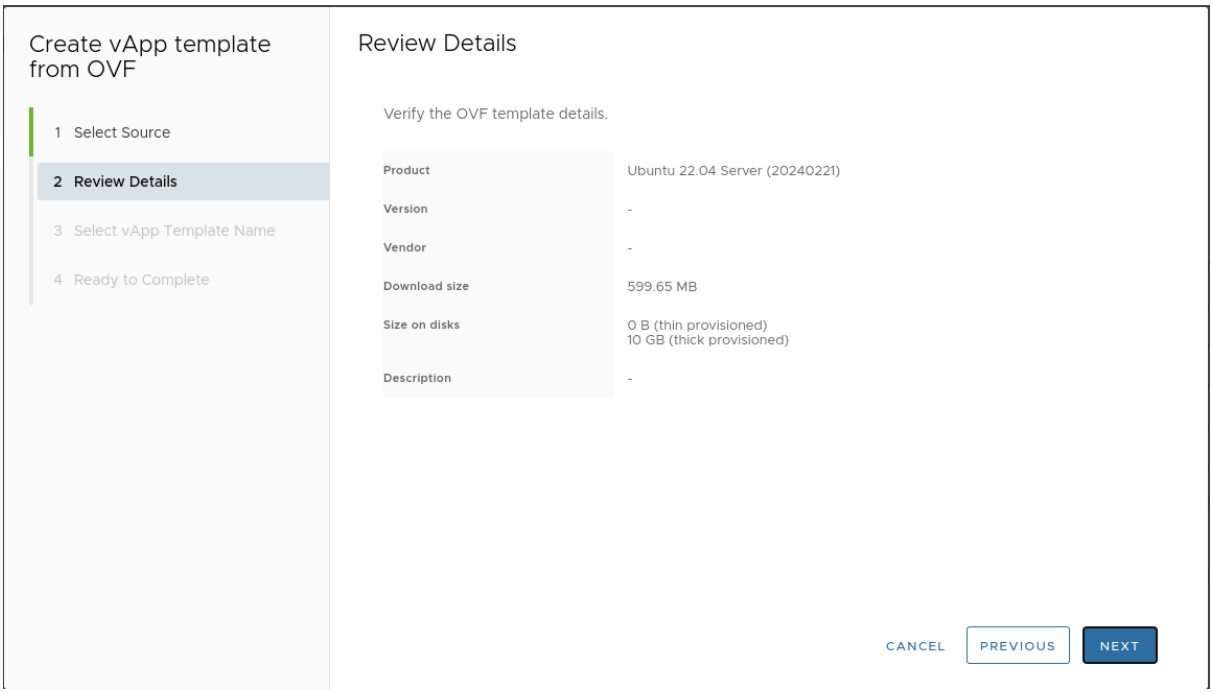


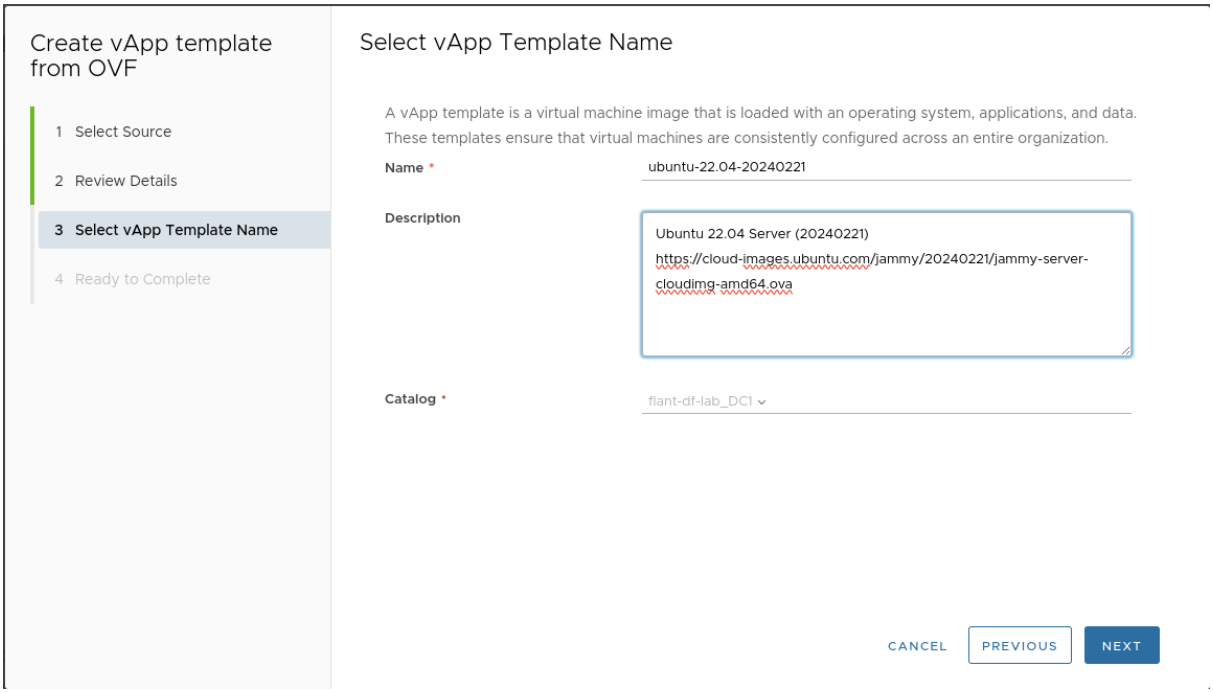
2. Go to “Libraries” → “Catalogs” → “Organization Catalog”:



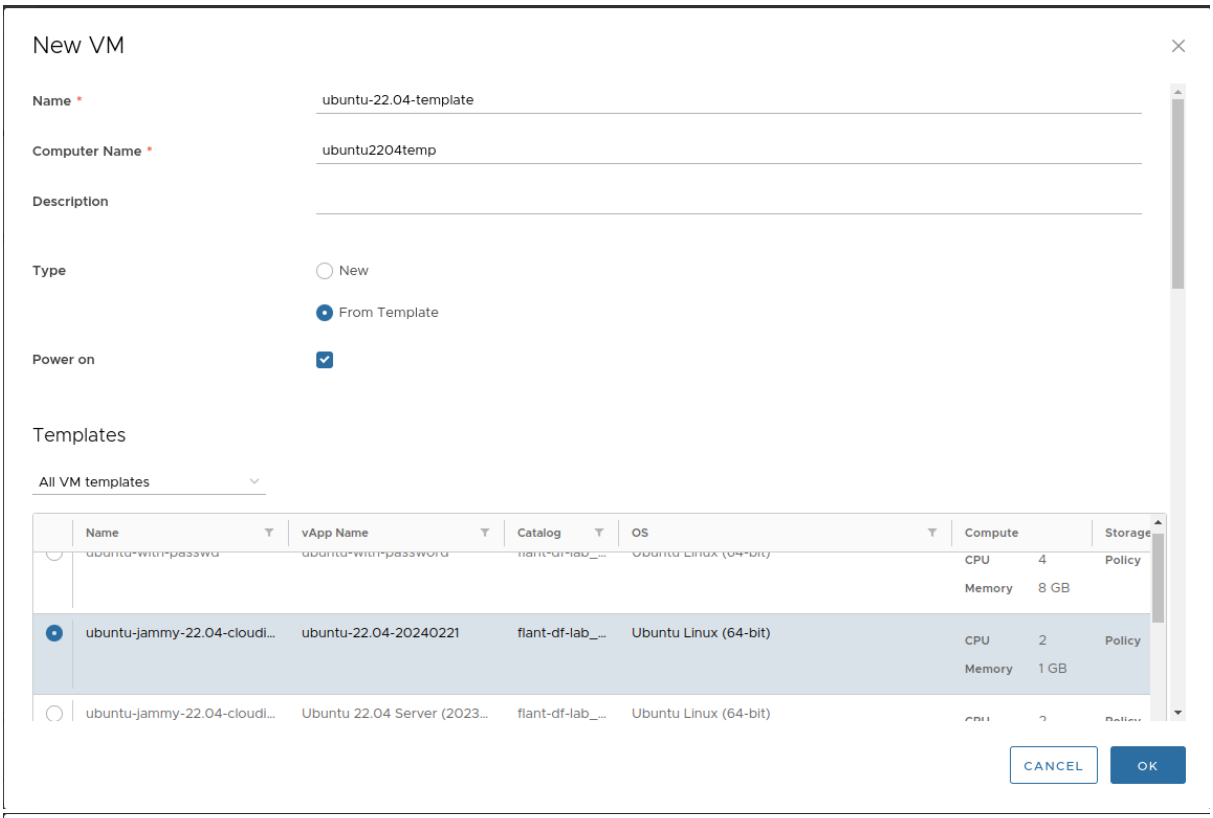
3. Upload the downloaded template to the catalog:

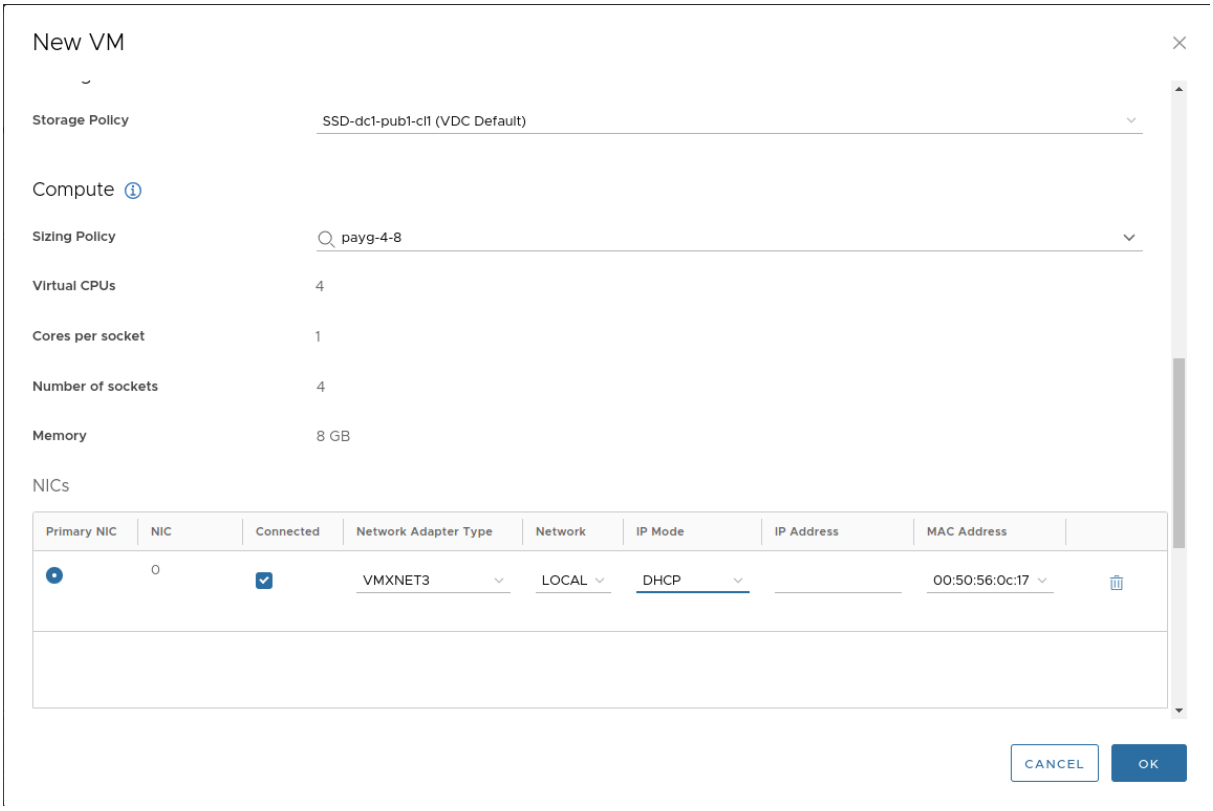




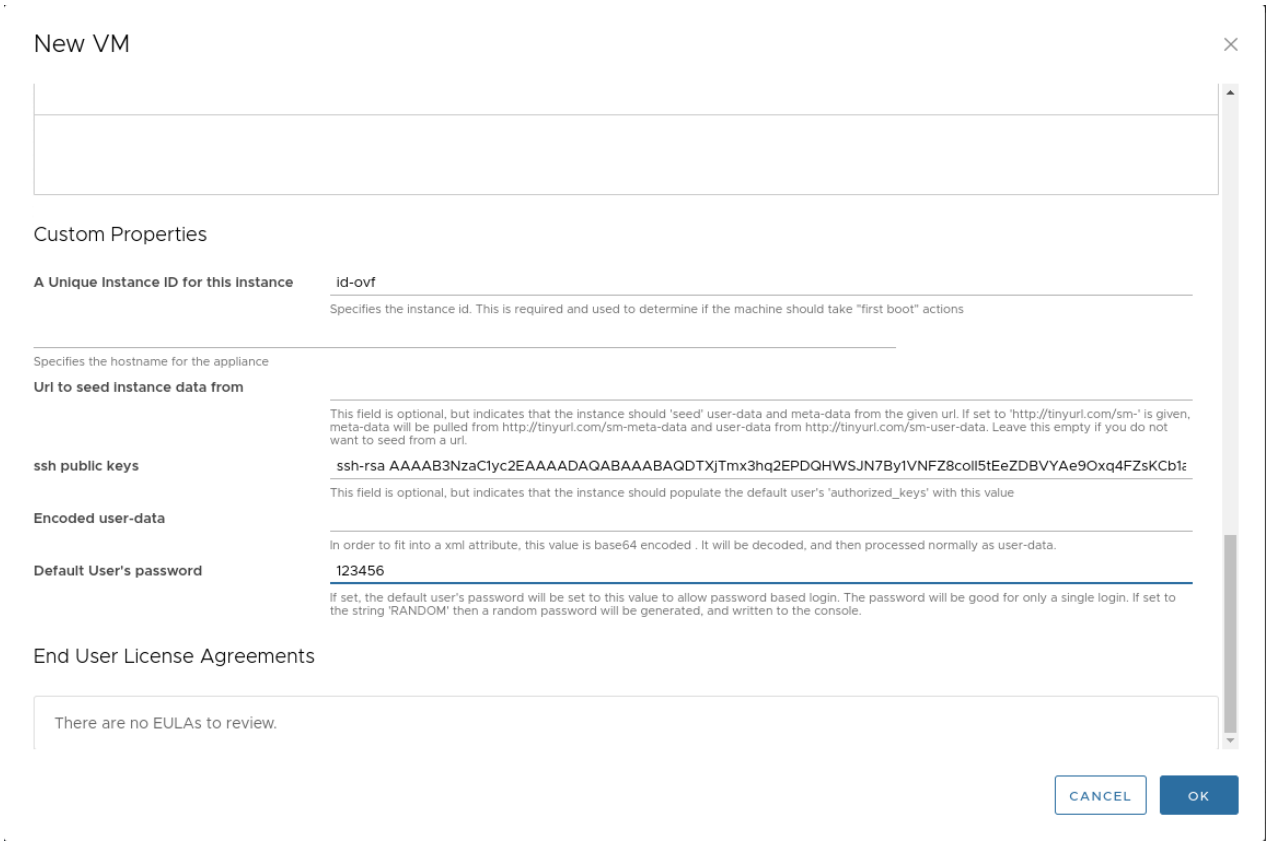


4. Create a VM from the template:



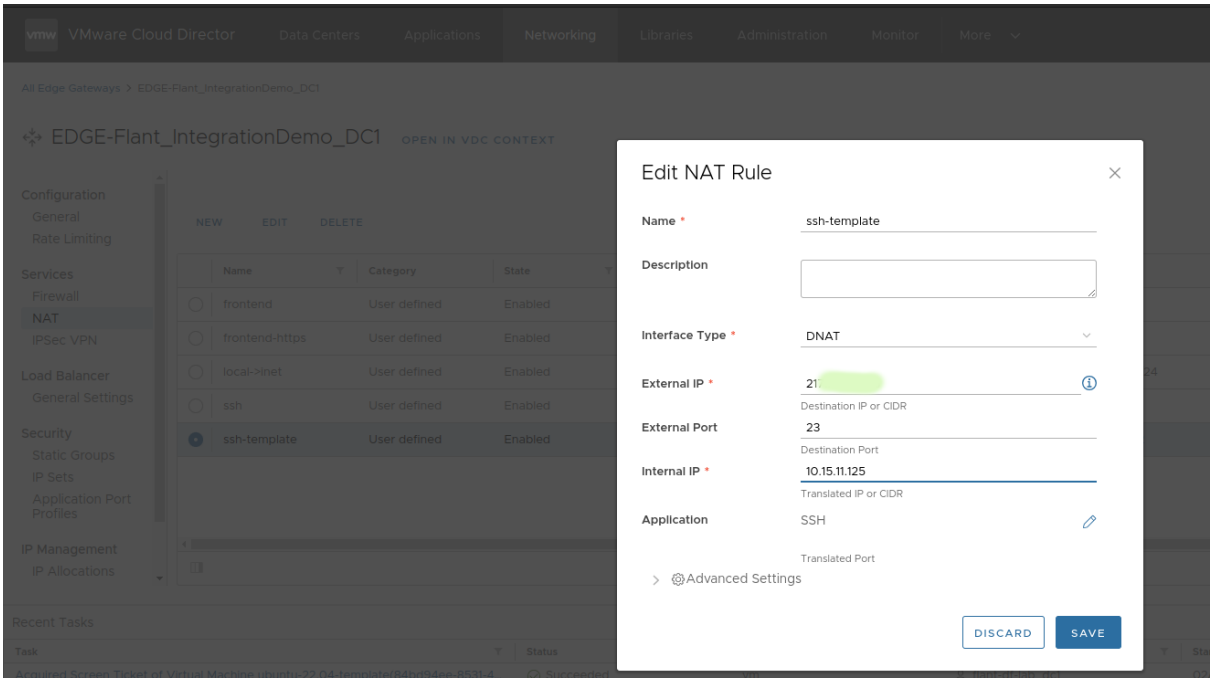


⚠ Set a default password and public key. These will be needed to log into the VM console.



To connect to the VM:

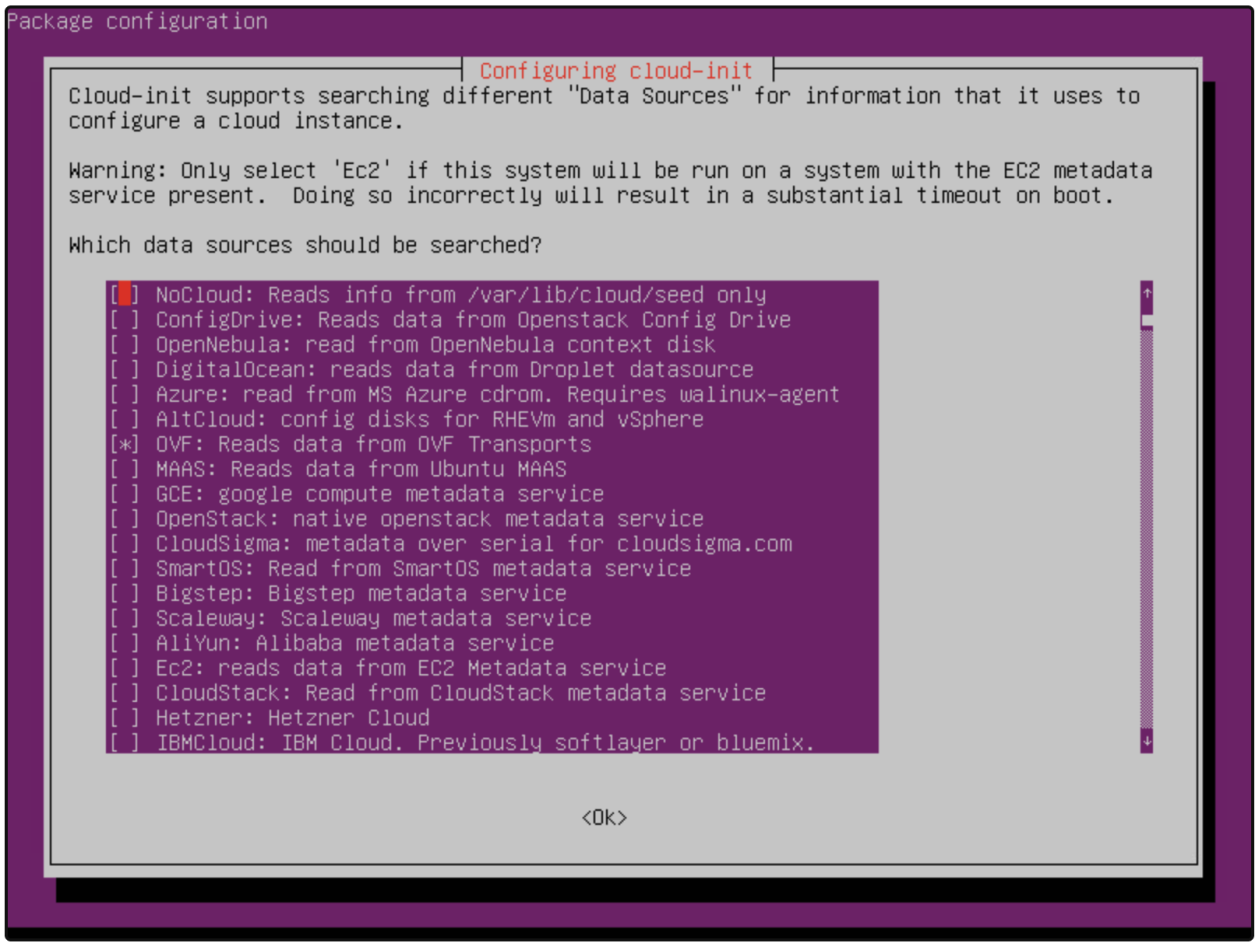
1. Start the VM.
2. Wait until it gets an IP address.
3. Forward port “22” to the VM:



Log into the VM via SSH and run:

```
rm /etc/netplan/99-netcfg-vmware.yaml
echo -e '\n[deployPkg]\nwait-cloudinit-timeout=1800\n' >> /etc/vmware-tools/tools.conf
echo 'disable_vmware_customization: true' > /etc/cloud/cloud.cfg.d/91_vmware_cust.cfg
dpkg-reconfigure cloud-init
```

In the dialog window, leave only the checkbox for `OVF: Reads data from OVF transports`. Disable all other options:



Make sure that the `datasource_list` parameter is specified in the `cloud-init` configuration. You can verify this using the following command:

```
cat /etc/cloud/cloud.cfg.d/90_dpkg.cfg
```

If the output is empty, execute the following command:

```
echo "datasource_list: [ OVF, VMware, None ]" > /etc/cloud/cloud.cfg.d/90_dpkg.cfg
```

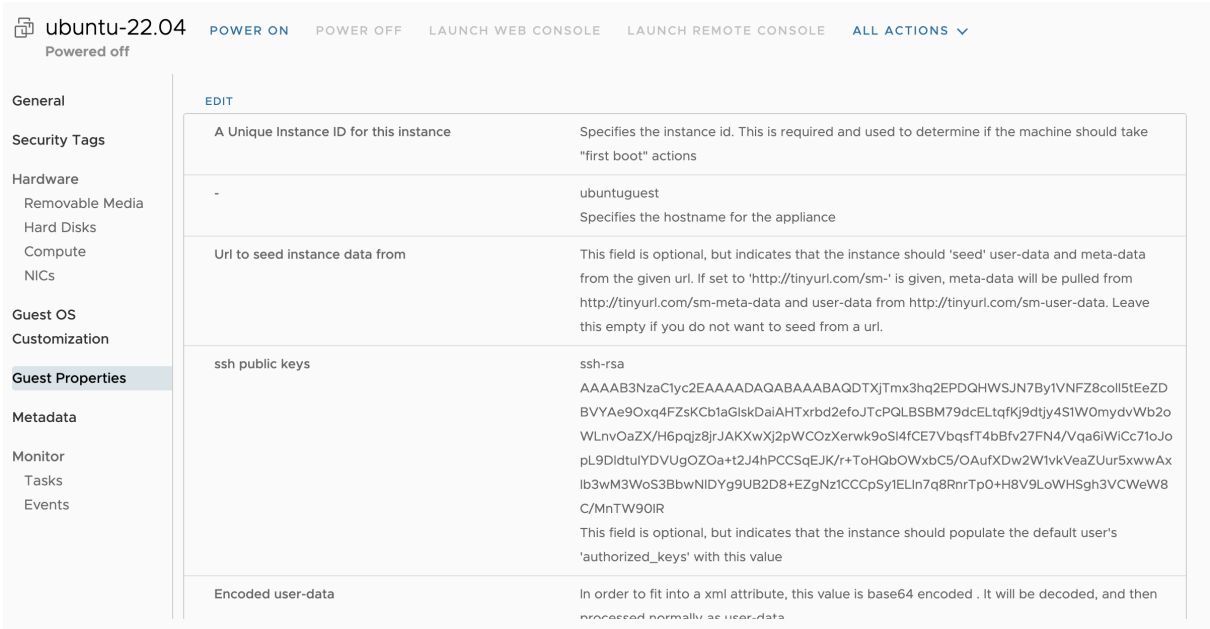
Run the remaining commands:

```
truncate -s 0 /etc/machine-id
rm /var/lib/dbus/machine-id
ln -s /etc/machine-id /var/lib/dbus/machine-id
cloud-init clean --logs --seed
passwd -d ubuntu
passwd -d root
rm /home/ubuntu/.ssh/authorized_keys
history -c

shutdown -P now
```

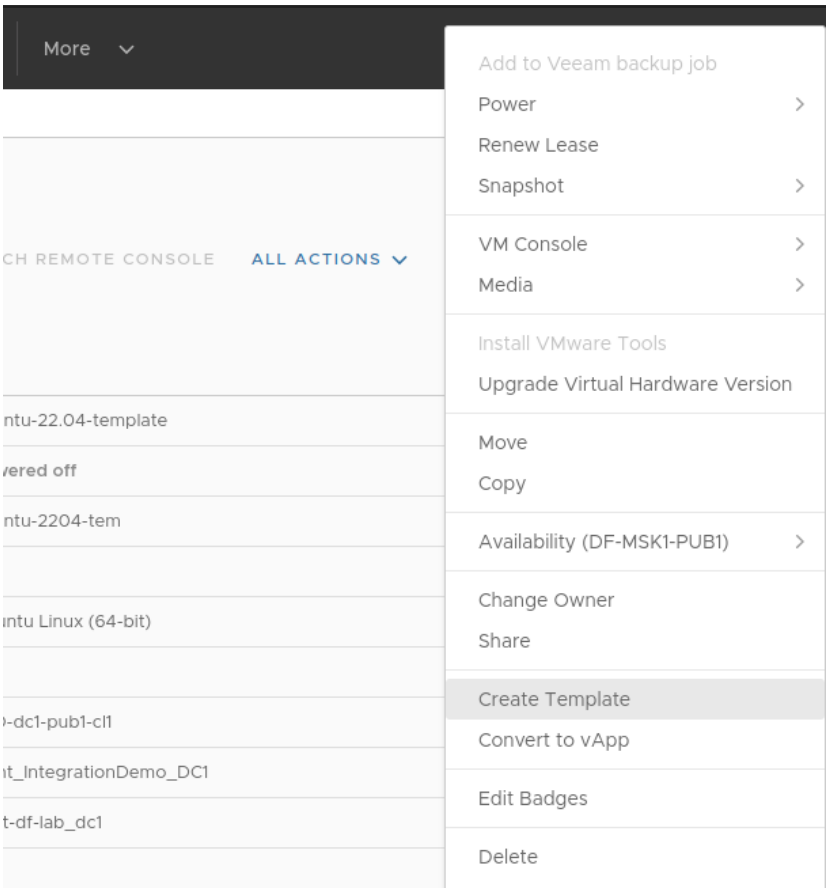
Configuring the template in VCD

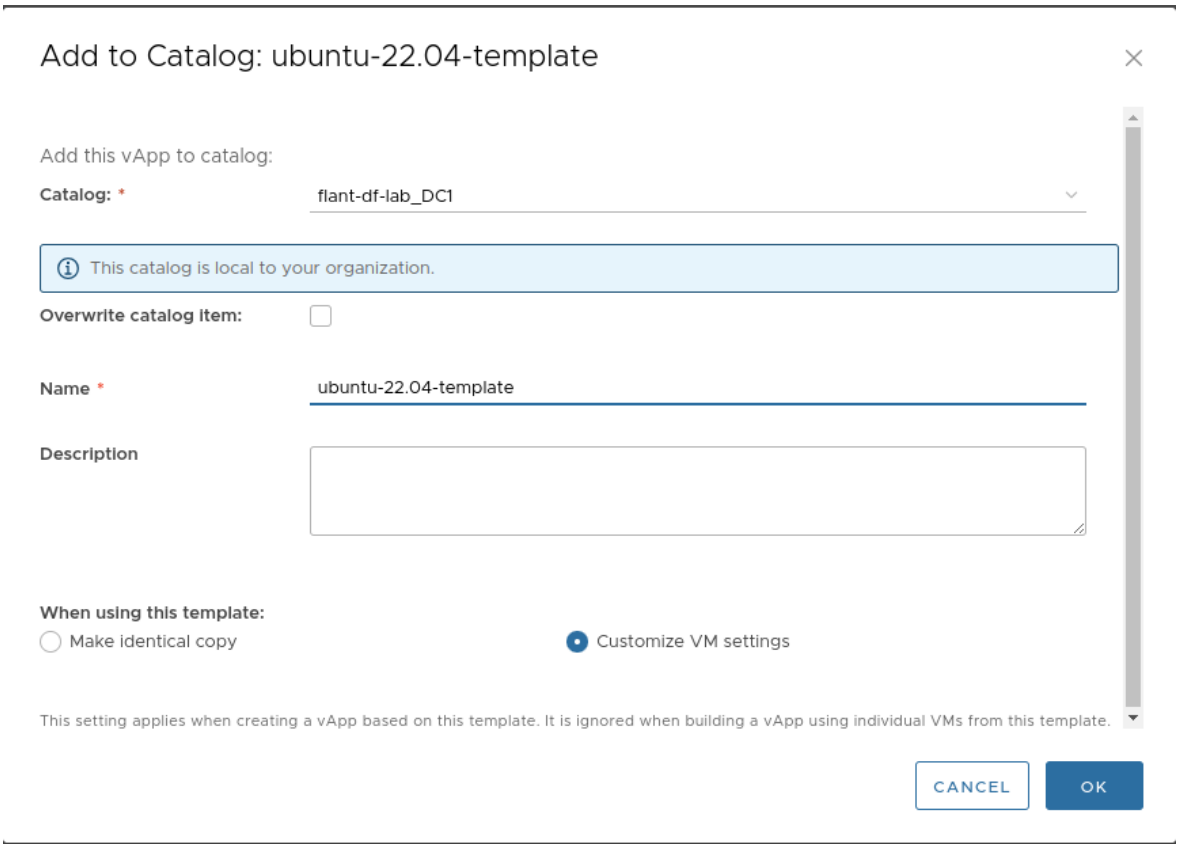
1. Power off the virtual machine and delete all filled “Guest Properties” fields:



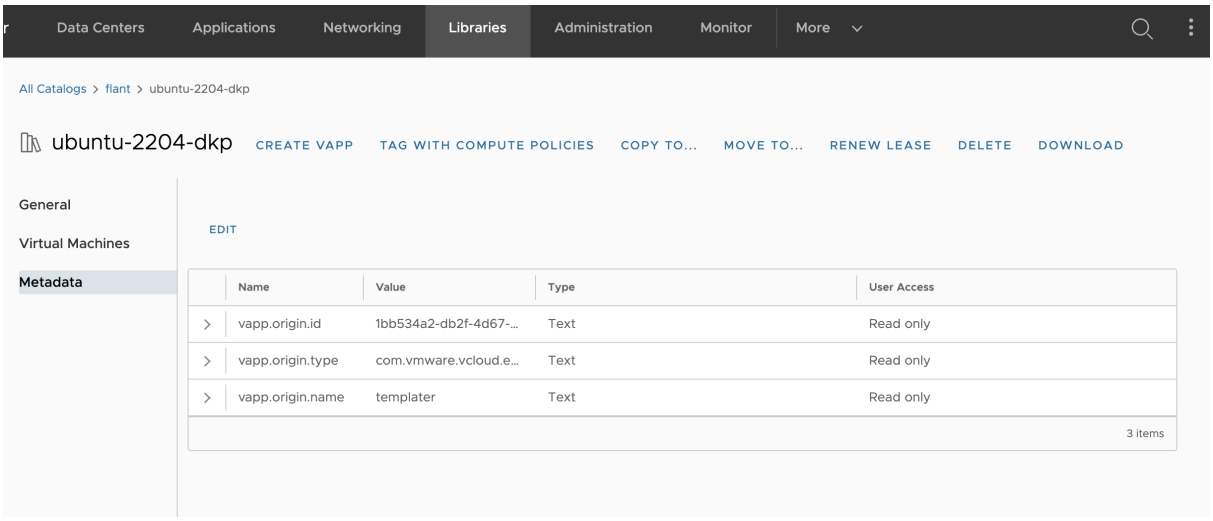


2. Create a virtual machine template:





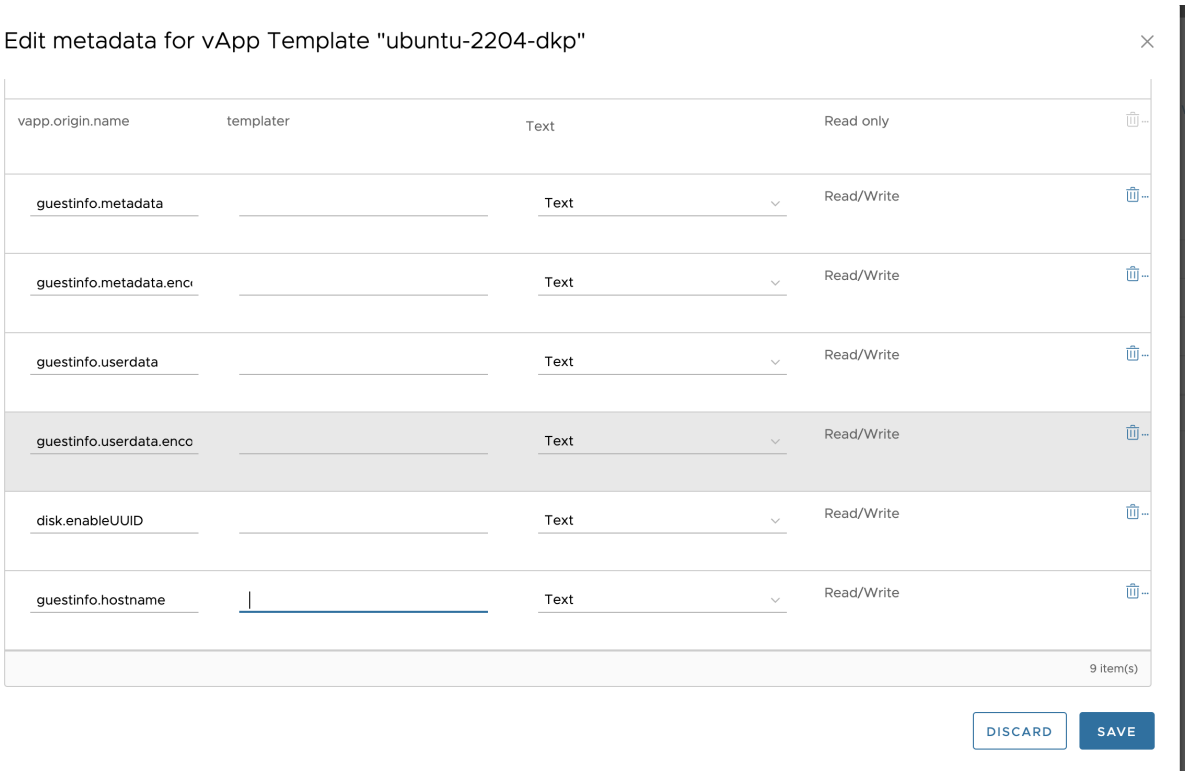
3. In the created template, go to the “Metadata” tab and add six fields:
   - `guestinfo.metadata`
   - `guestinfo.metadata.encoding`
   - `guestinfo.userdata`
   - `guestinfo.userdata.encoding`
   - `disk.enableUUID`
   - `guestinfo.hostname`



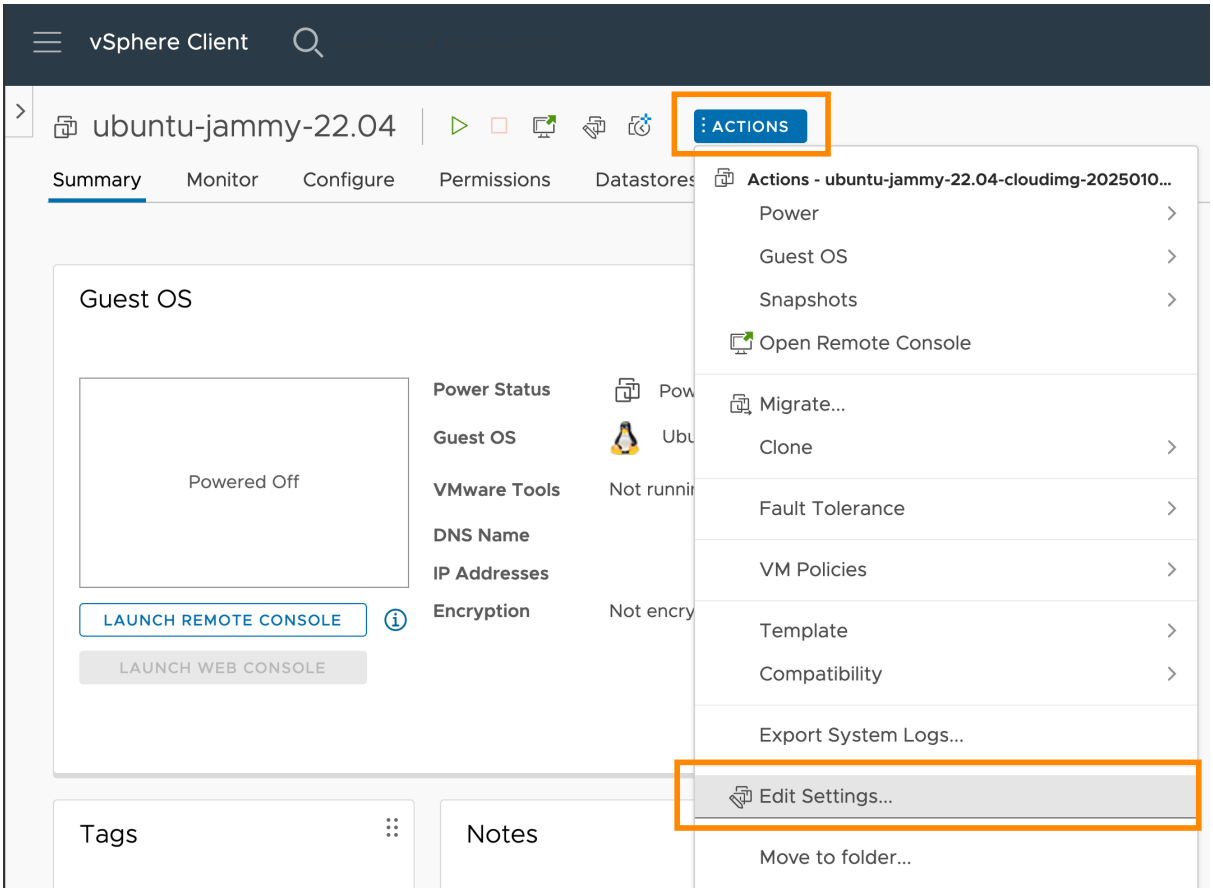
For **each** field in the add/edit form, set the following:

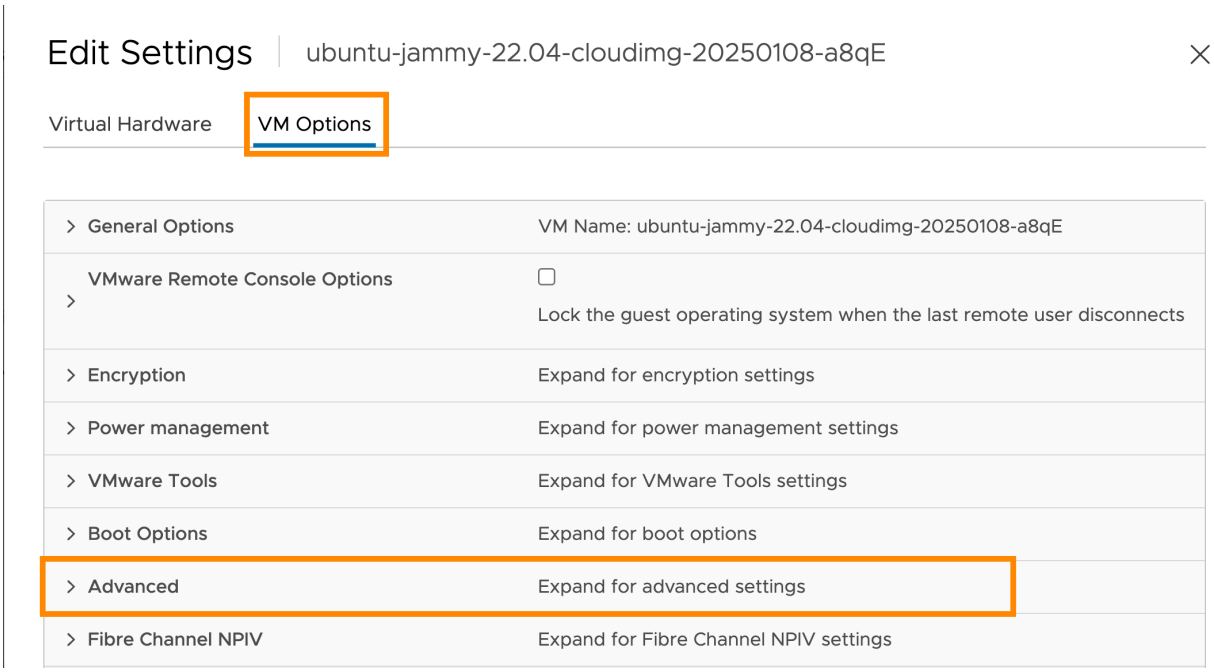
- “Type”: `Text` (text value)
- “User access”: `Read/Write`
- “Value”: A single space

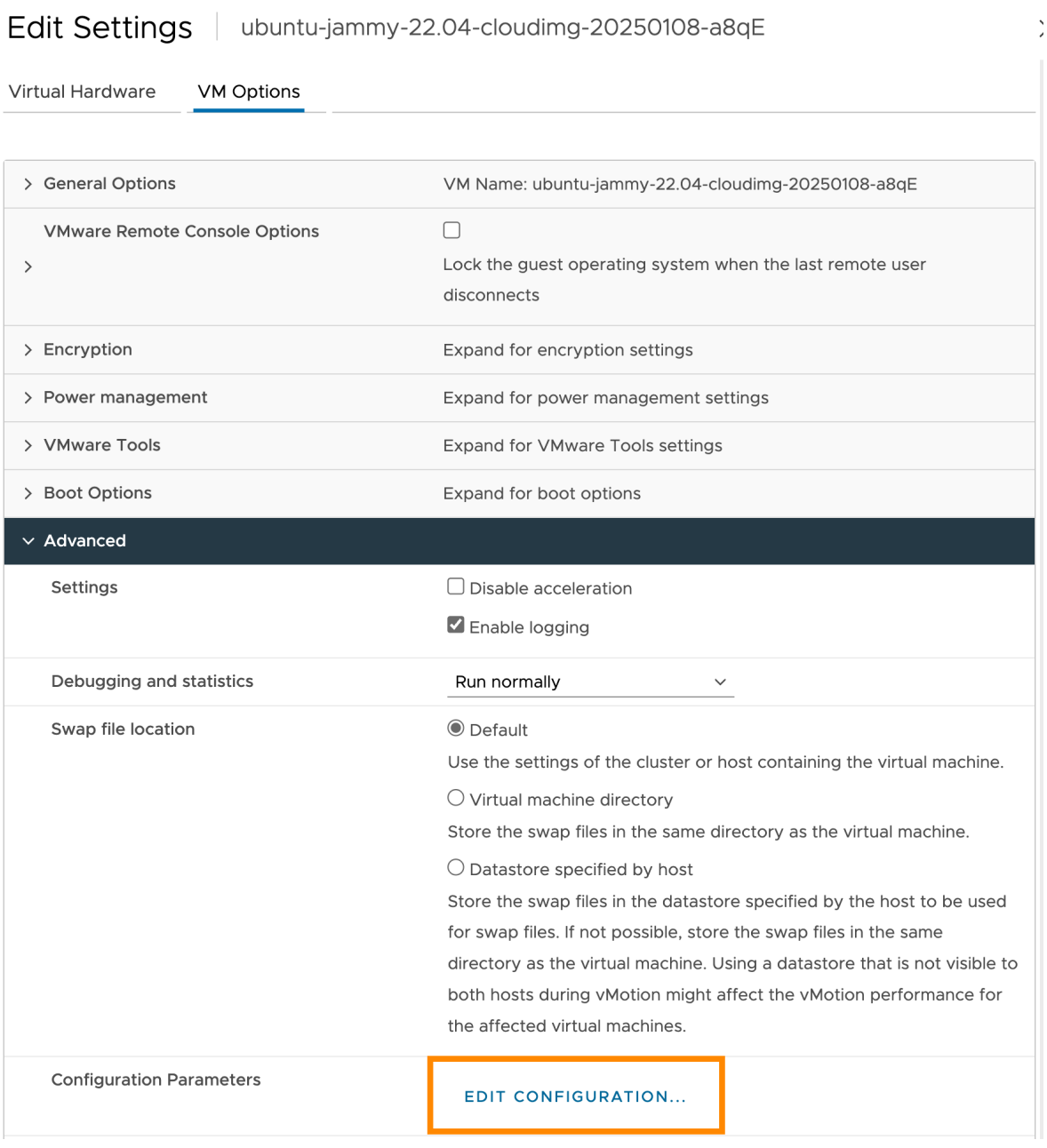
> **i** The VCD UI may not save metadata with an empty value. A single space is used as a placeholder and does not affect functionality. The actual values will be populated automatically when virtual machines are created.

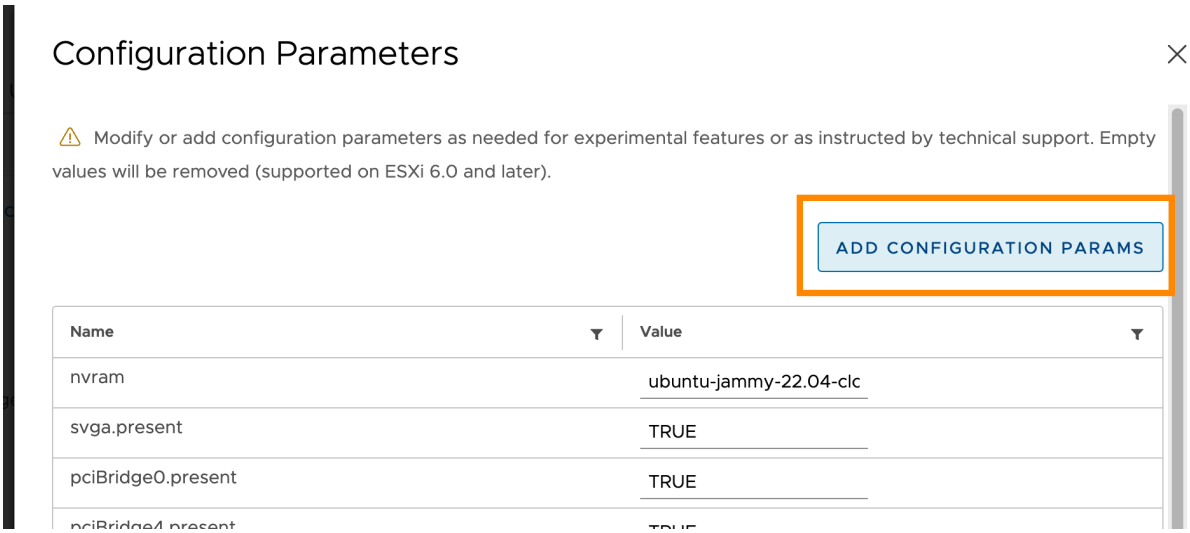


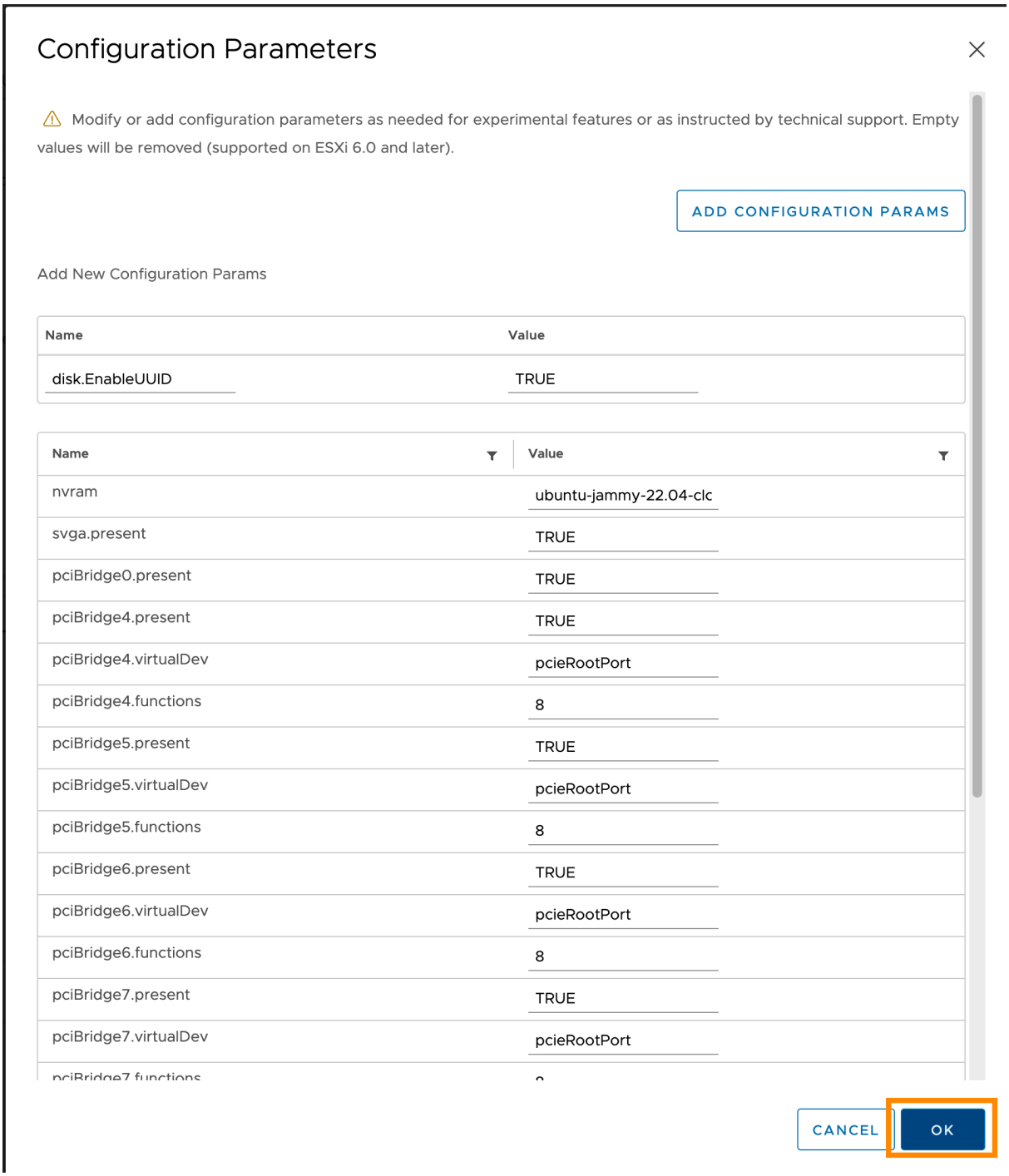
4. In the vCenter management panel, enable the `disk.EnableUUID` parameter for the template:




Edit Settings
ubuntu-jammy-22.04-cloudimg-20250108-a8qE
Virtual Hardware
VM Options
General Options
VM Name: ubuntu-jammy-22.04-cloudimg-20250108-a8qE
VMware Remote Console Options
Lock the guest operating system when the last remote user disconnects
Encryption
Expand for encryption settings
Power management
Expand for power management settings
VMware Tools
Expand for VMware Tools settings
Boot Options
Expand for boot options
Advanced
Expand for advanced settings
Fibre Channel NPIV
Expand for Fibre Channel NPIV settings


Edit Settings
ubuntu-jammy-22.04-cloudimg-20250108-a8qE
Virtual Hardware
VM Options
General Options
VM Name: ubuntu-jammy-22.04-cloudimg-20250108-a8qE
VMware Remote Console Options
Lock the guest operating system when the last remote user disconnects
Encryption
Expand for encryption settings
Power management
Expand for power management settings
VMware Tools
Expand for VMware Tools settings
Boot Options
Expand for boot options
Advanced
Settings
Disable acceleration
Enable logging
Debugging and statistics
Run normally
Swap file location
Default
Use the settings of the cluster or host containing the virtual machine.
Virtual machine directory
Store the swap files in the same directory as the virtual machine.
Datastore specified by host
Store the swap files in the datastore specified by the host to be used for swap files. If not possible, store the swap files in the same directory as the virtual machine. Using a datastore that is not visible to both hosts during vMotion might affect the vMotion performance for the affected virtual machines.
Configuration Parameters
EDIT CONFIGURATION...





Storage usage

- VCD supports CSI. Disks are created as VCD Independent Disks.
- The guest property `disk.EnableUUID` must be enabled for the VM templates in use.

- Deckhouse Kubernetes Platform supports disk resizing starting from version v1.59.1.

Using the LoadBalancer

- DKP components support `Service` resources of type `LoadBalancer` when deployed on VMware Cloud Director (VCD).
- VMware NSX Advanced Load Balancer (ALB or Avi Networks) is used as the load balancer.
- Support is available **only** when using the `NSX-T` network virtualization platform.
- The load balancer functionality must be enabled on the Edge Gateway by your VCD provider. You can verify this under Edge Gateway → Load Balancer → General Settings — the `State` parameter must be `Active`.
- If the load balancer was enabled after the DKP cluster was successfully created, the components will automatically pick up the changes within an hour (no additional actions are required).
- For each open port, a Pool + Virtual Service pair is created.
- If a firewall is in place, you must create an allow rule for the load balancer's external IP address and the corresponding ports.

Additional resources

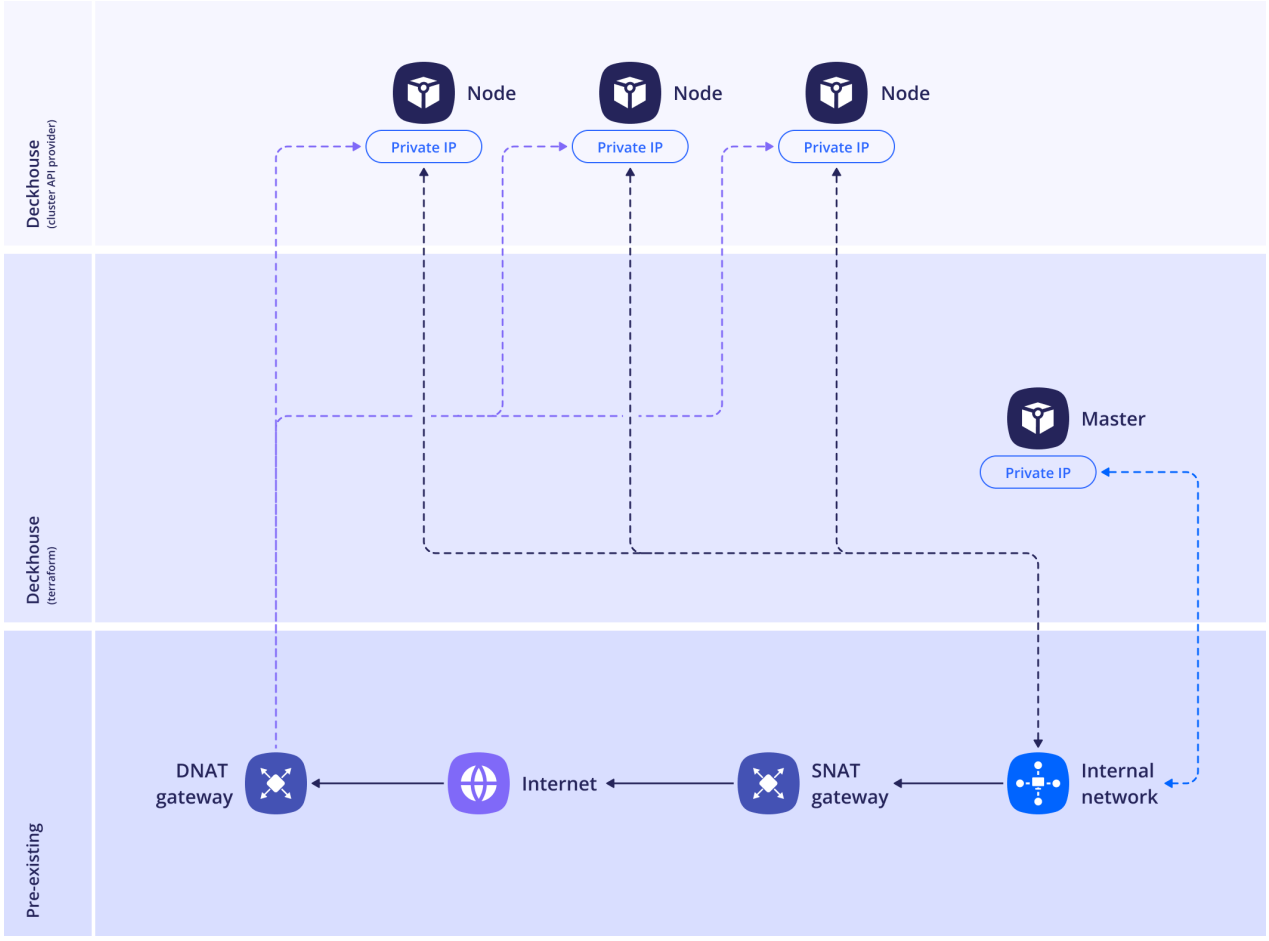
- Module `cert-manager` documentation
- Module `metallb` documentation

Layouts and configuration in VMware Cloud Director

Layouts

Deckhouse Kubernetes Platform supports two layouts for deploying resources in VCD.

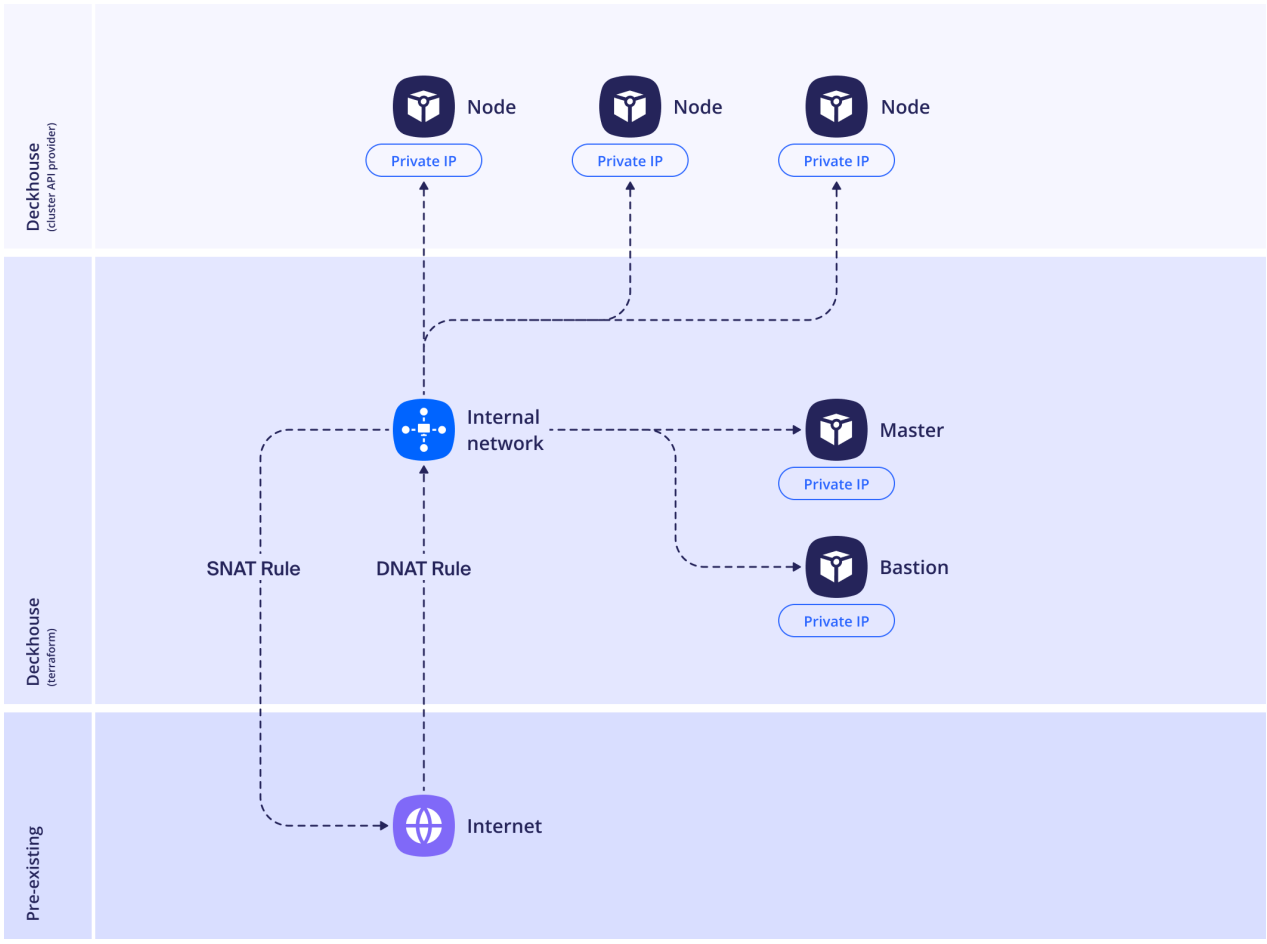
Standard



Example layout configuration:

```
---
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: Standard
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetwork: internal
    mainNetworkIPAddresses:
    - 192.168.199.2
```

WithNAT



When using this layout, you must confirm the type of network virtualization platform with your administrator and specify it in the `edgeGateway.type` parameter. Two options are supported: `NSX-T` and `NSX-V`.

To ensure administrative access to the cluster nodes, a bastion is deployed. The parameters for its configuration are described in the `bastion` section.

If the Edge Gateway is based on `NSX-T`, a DHCP server will be automatically enabled for the created node network. It will assign IP addresses starting from the 30th address in the subnet up to the penultimate one (before the broadcast address). The starting address of the DHCP pool can be changed using the `internalNetworkDHCPPoolStartAddress` parameter.

If `NSX-V` is used, DHCP must be configured manually. Otherwise, nodes waiting to obtain an IP address via DHCP will not be able to receive one.

⚠ It is not recommended to use dynamic addressing for the first master node together with `NSX-V`.

This layout assumes the automatic creation of the following NAT rules:

- **SNAT** — translation of internal node network addresses to the external address specified in `edgeGateway.externalIP`.
- **DNAT** — translating the external address and port, specified in the `edgeGateway.externalIP` and `edgeGateway.externalPort` properties, respectively, to the internal address of the bastion instance on port 22 using the TCP protocol for administrative access to the nodes via SSH.

⚠ If the Edge Gateway is powered by `NSX-V`, you must specify the name and type of the network to which the rule will be bound in the `edgeGateway.NSX-V.externalNetworkName` and `edgeGateway.NSX-V.externalNetworkType` properties. Usually, this is the network connected to the Edge Gateway in the `Gateway Interfaces` section and having an external IP address.

It is also possible to create firewall rules using the `createDefaultFirewallRules` property.

⚠ If the Edge Gateway is powered by `NSX-T`, existing Edge Gateway rules will be overwritten. This option is intended for scenarios where only one cluster is deployed on the Edge Gateway.

The following rules will be created:

- Allow all outgoing traffic;
- Allow incoming TCP traffic on port 22 for SSH connections to cluster nodes;
- Allow all incoming ICMP traffic;
- Allow incoming TCP and UDP traffic on ports 30000–32767 for `NodePort` services.

Example configuration for the WithNAT layout using `NSX-T`:

```
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: WithNAT
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
internalNetworkDNSServers:
  - 77.88.8.8
  - 1.1.1.1
mainNetwork: internal
bastion:
  instanceClass:
    rootDiskSizeGb: 30
    sizingPolicy: 2cpu1mem
    template: "catalog/Ubuntu 22.04 Server"
    storageProfile: Fast vHDD
    mainNetworkIPAddress: 10.1.4.10
edgeGateway:
  name: "edge-gateway-01"
  type: "NSX-T"
  externalIP: 10.0.0.1
  externalPort: 10022
createDefaultFirewallRules: false
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
    - 192.168.199.2
```

Example configuration for the WithNAT layout using `NSX-V`:

```
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: WithNAT
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
internalNetworkDNSServers:
  - 77.88.8.8
  - 1.1.1.1
mainNetwork: internal
bastion:
  instanceClass:
    rootDiskSizeGb: 30
    sizingPolicy: 2cpu1mem
    template: "catalog/Ubuntu 22.04 Server"
    storageProfile: Fast vHDD
    mainNetworkIPAddress: 10.1.4.10
edgeGateway:
  name: "edge-gateway-01"
  type: "NSX-V"
  externalIP: 10.0.0.1
  externalPort: 10022
  NSX-V:
    externalNetworkName: external
    externalNetworkType: ext
createDefaultFirewallRules: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
    - 192.168.199.2
```

Configuration

Integration is carried out using the VCDClusterConfiguration resource, which describes the configuration of the cloud cluster in VCD and is used by the virtualization system if the cluster's control plane is hosted there. The DKP module responsible for integration is automatically configured based on the selected deployment layout.

To modify the configuration in a running cluster, execute the following command:

```
d8 -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller edit provider-cluster-configuration
```

> **i** After changing the node parameters, you must run the `dhctl converge` command for the changes to take effect.

Example configuration:

```
apiVersion: deckhouse.io/v1
kind: VCDClusterConfiguration
sshPublicKey: "<SSH_PUBLIC_KEY>"
organization: My_Org
virtualDataCenter: My_Org
virtualApplicationName: Cloud
mainNetwork: internal
layout: Standard
internalNetworkCIDR: 172.16.2.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    template: Templates/ubuntu-focal-20.04
    sizingPolicy: 4cpu8ram
    rootDiskSizeGb: 20
    etcdDiskSizeGb: 20
    storageProfile: nvme
nodeGroups:
  - name: worker
    replicas: 1
    instanceClass:
      template: Org/Templates/ubuntu-focal-20.04
      sizingPolicy: 16cpu32ram
      storageProfile: ssd
provider:
  server: "<SERVER>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  insecure: true
```

The number of nodes to be provisioned and their parameters are defined in the NodeGroup custom resource, where you also specify the name of the instance class used for that group (the `cloudInstances.classReference` parameter). For the VCD cloud provider, the instance class is a custom resource called VCDInstanceClass, which contains the specific configuration of the VMs.

The following is the example configuration of VCDInstanceClass for ephemeral nodes of the VMware Cloud Director cloud provider.

Example configuration of VCDInstanceClass

```
apiVersion: deckhouse.io/v1
kind: VCDInstanceClass
metadata:
  name: test
spec:
  rootDiskSizeGb: 90
  sizingPolicy: payg-4-8
  storageProfile: SSD-dc1-pub1-cl1
  template: MyOrg/Linux/ubuntu2204-cloud-ova
```

Storage

A StorageClass is automatically created for each Datastore and DatastoreCluster in the zone (or zones).

You can set the name of StorageClass that will be used in the cluster by default (the `default` parameter), and filter out the unnecessary StorageClasses (the `exclude` parameter).

CSI

By default, the storage subsystem uses CNS volumes capable of on-the-fly resizing. FCD volumes are also supported, but only in the legacy mode. You can set this via the `compatibilityFlag` parameter.

Additional resources

- Module `cloud-provider-vcd` provider configuration
- Module `cloud-provider-vcd` custom resources
- Module `node-manager` custom resources

Connection and authorization in VMware vSphere

Requirements

The following prerequisites must be met for Deckhouse Kubernetes Platform to work correctly with VMware vSphere:

- Access to vCenter;

- A user account with the required set of privileges;
- Tags and tag categories created in vSphere;
- Networks with DHCP and Internet access;
- Shared Datastore resources available on all ESXi hosts in use;
- vSphere version `7.x` or `8.x` with support for `Online volume expansion`;
- vCenter accessible from inside the cluster from the master nodes;
- A configured Datacenter that includes the following objects:
  1. VirtualMachine template:
     - The virtual machine image must use `Virtual machines with hardware version 15 or later` — this is required for online resize support.
     - The image must include the `open-vm-tools`, `cloud-init`, and `cloud-init-vmware-guestinfo` packages if the `cloud-init` version is lower than `21.3`.
  2. Network:
     - The network must be available on all ESXi hosts where virtual machines are planned to be created.
  3. Datastore (one or more):
     - The Datastore must be connected to all ESXi hosts where virtual machines are planned to be created.
     - A tag from the category specified in the `zoneTagCategory` parameter must be assigned to the Datastore (default: `k8s-zone`). This tag defines the zone.
     - All Cluster objects within the same zone must have access to all Datastore objects in that zone.
  4. Cluster:
     - All ESXi hosts in use must be added to the Cluster.
     - A tag from the category specified in the `zoneTagCategory` parameter must be assigned to the Cluster (default: `k8s-zone`). This tag defines the zone.
  5. Folder for the virtual machines being created:
     - This parameter is optional.
     - By default, the root virtual machine folder is used.
  6. Role:
     - The role must include the required set of privileges.
  7. User:
     - The user must be assigned the role specified in the previous item.
- A tag from the category specified in the `regionTagCategory` parameter must be assigned to the created Datacenter (default: `k8s-region`). This tag defines the region.

VM image requirements

To create a VM template (Template), it is recommended to use a ready-made cloud image/OVA file provided by the OS vendor:

- **Ubuntu**
- **Debian**
- **CentOS**
- **Rocky Linux** (section *Generic Cloud / OpenStack*)

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Preparing the virtual machine image

> ⚠ Disable VMware Guest OS Customization (and any vApp/OS customization mechanisms, if applicable in your setup) for the template and the cluster virtual machines. DKP performs the initial node configuration via `cloud-init` (VMware GuestInfo datasource). Enabled customization can conflict with `cloud-init` and lead to incorrect node initialization.

1. Install the required packages:

   If you use `cloud-init` version lower than 21.3 (VMware GuestInfo support is required):

   ```
   sudo apt-get update
   sudo apt-get install -y open-vm-tools cloud-init cloud-init-vmware-guestinfo
   ```

   If you use `cloud-init` version 21.3 or higher:

   ```
   sudo apt-get update
   sudo apt-get install -y open-vm-tools cloud-init
   ```

2. Verify that the `disable_vmware_customization: false` parameter is set in `/etc/cloud/cloud.cfg`.
3. Make sure the `default_user` parameter is specified in `/etc/cloud/cloud.cfg`. It is required to add an SSH key when the VM starts.
4. Add the VMware GuestInfo datasource — create `/etc/cloud/cloud.cfg.d/99-DataSourceVMwareGuestInfo.cfg`:

   ```
   datasource:
     VMware:
       vmware_cust_file_max_wait: 10
   ```

5. Before creating the VM template, reset the `cloud-init` identifiers and state using the following commands:

```
truncate -s 0 /etc/machine-id &&
rm /var/lib/dbus/machine-id &&
ln -s /etc/machine-id /var/lib/dbus/machine-id
```

6. Clear `cloud-init` event logs:

```
cloud-init clean --logs --seed
```

⚠ After the virtual machine starts, the following services related to the packages installed during `cloud-init` preparation must be running:

- `cloud-config.service`,
- `cloud-final.service`,
- `cloud-init.service`.

To ensure that the services are enabled, use the command:

```
systemctl is-enabled cloud-config.service cloud-init.service cloud-final.service
```

Example output for enabled services:

```
enabled
enabled
enabled
```

i DKP creates VM disks of type `eagerZeroedThick`, but the type of disks of created VMs may be changed without notification according to the `VM Storage Policy` settings in vSphere.
For more details, see the documentation.

i DKP uses the `ens192` interface as the default interface for VMs in vSphere. Therefore, when using static IP addresses in `mainNetwork`, you must create an interface named `ens192` in the OS image as the default interface.

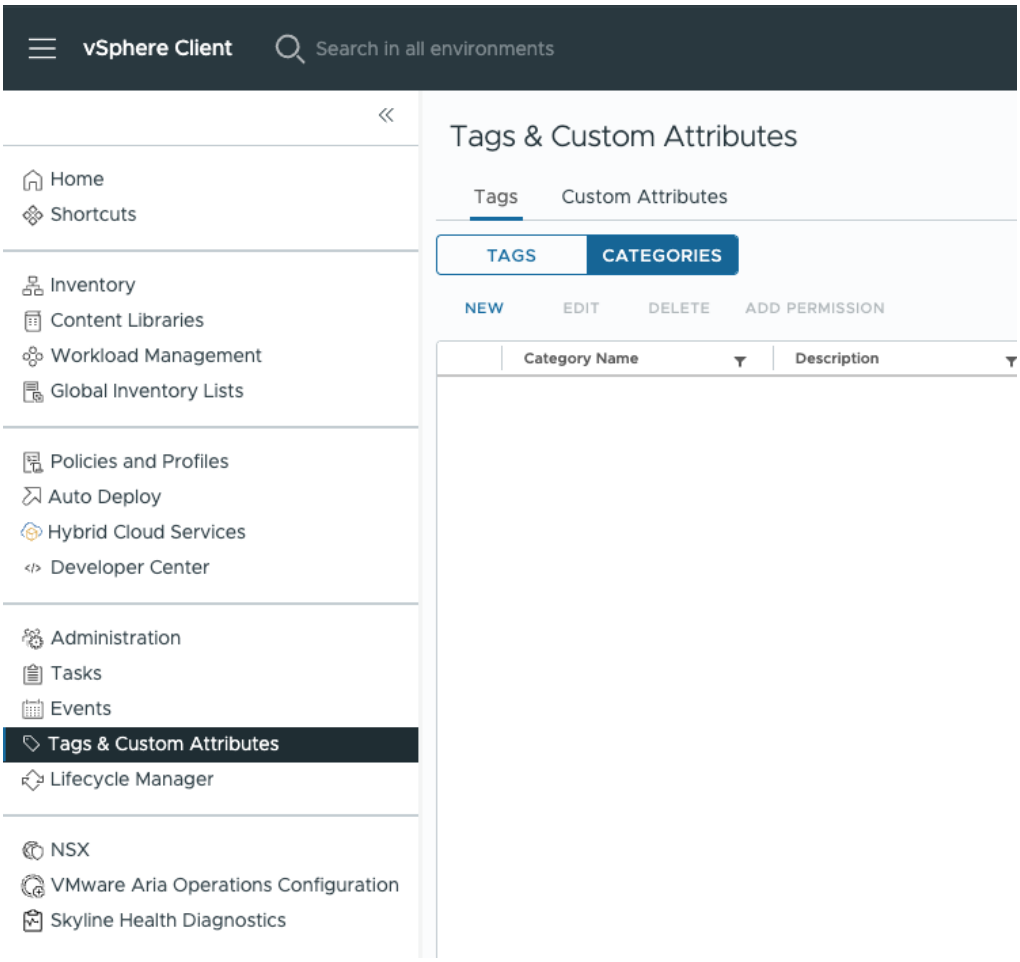
vSphere configuration

Configuration in vSphere Client

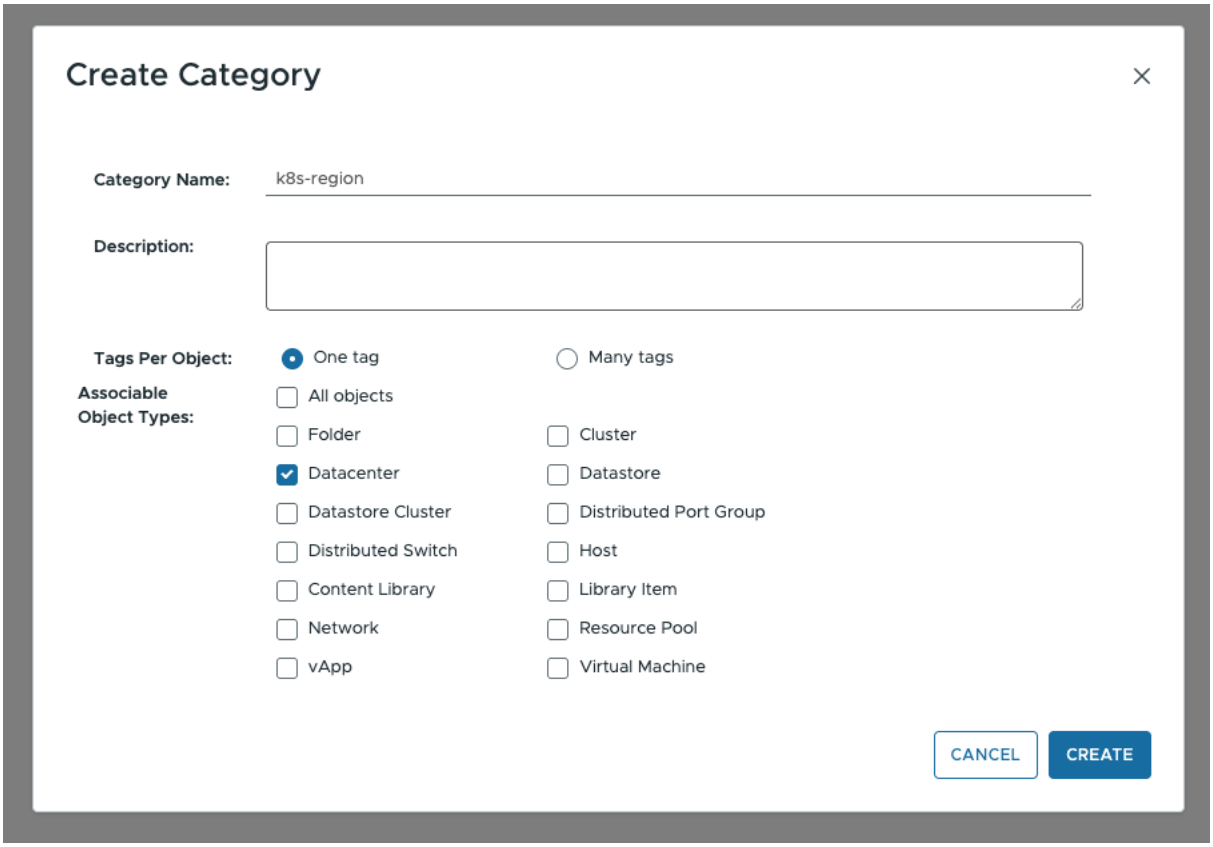
Creating tags and tag categories in vSphere Client

VMware vSphere does not have built-in concepts of a “region” or a “zone”. In vSphere, a Datacenter acts as a region, and a Cluster acts as a zone. Tags are used to establish this mapping.

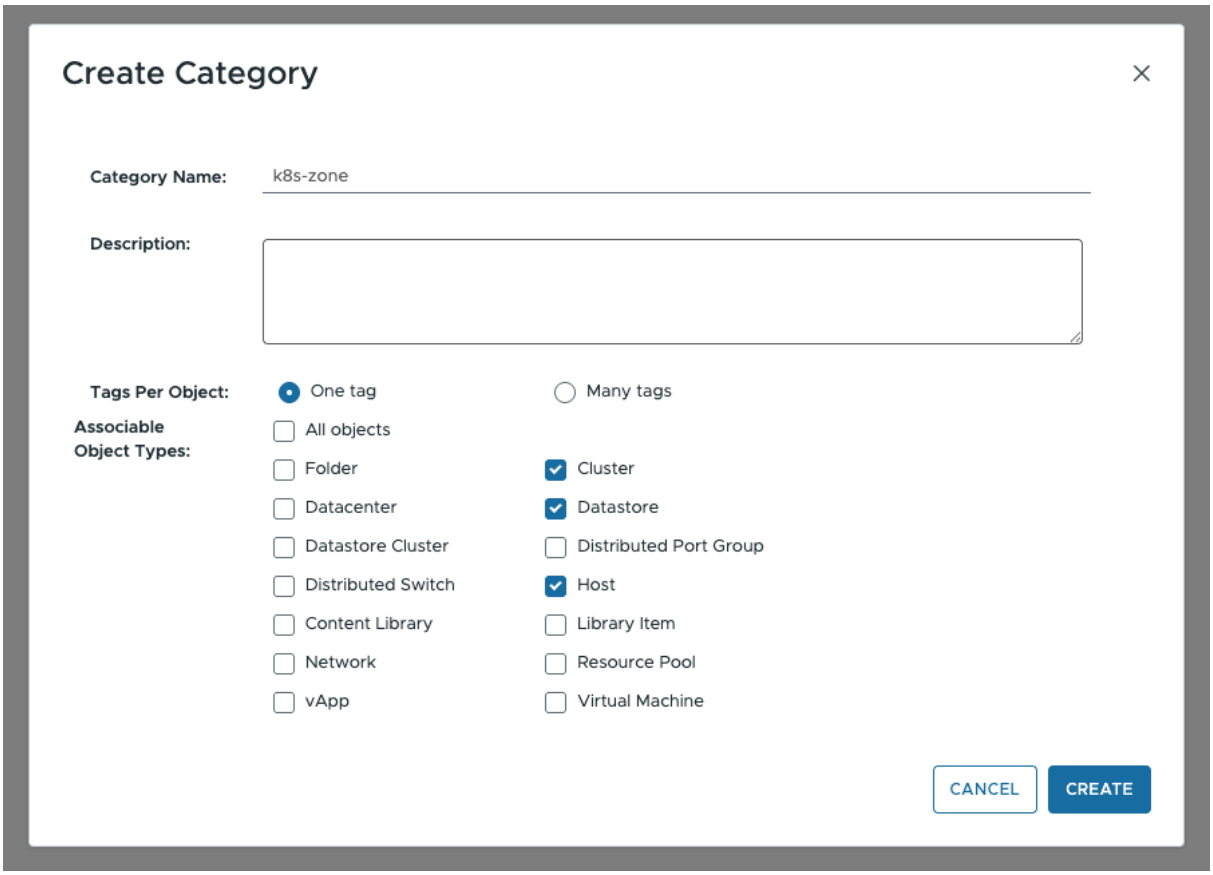
1. Open vSphere Client and go to “Menu” → “Tags & Custom Attributes” → “Tags”.



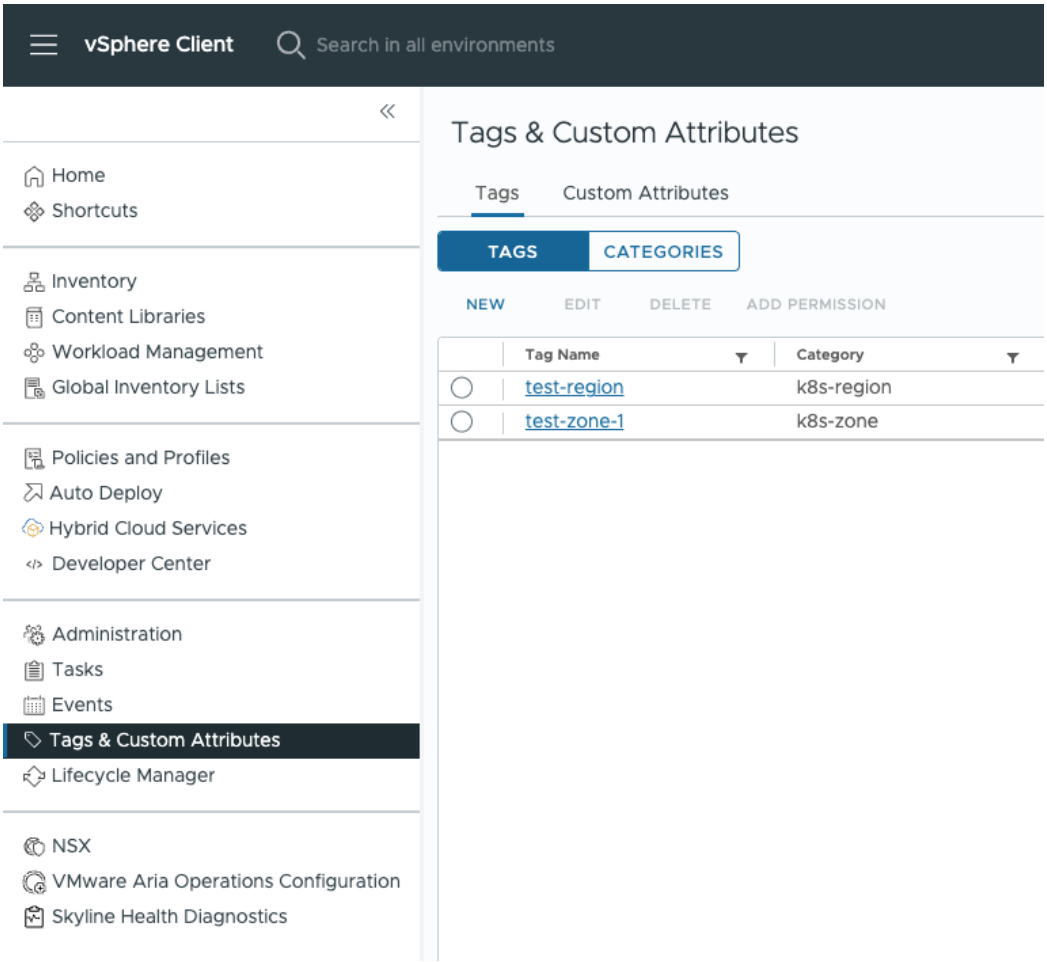
2. Open the “Categories” tab and click “NEW”. Create a category for regions (for example, `k8s-region`): set “Tags Per Object” to “One tag” and specify the applicable object types, including Datacenter.



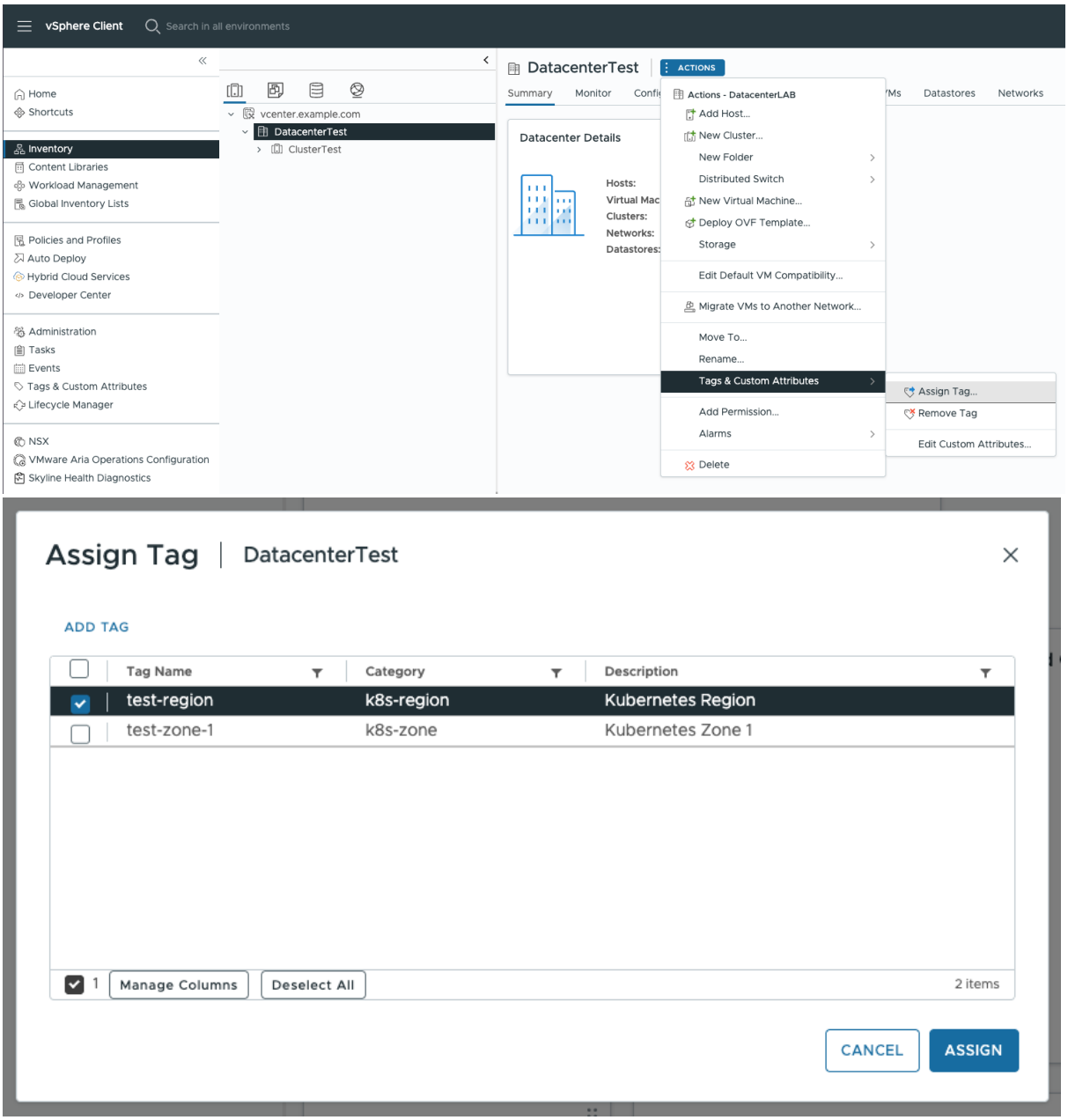
3. Create a second category for zones (for example, `k8s-zone`) with the object types Host, Cluster, and Datastore.



4. Go to the “Tags” tab and create at least one tag in the region category and one tag in the zone category (for example, `test-region`, `test-zone-1`).



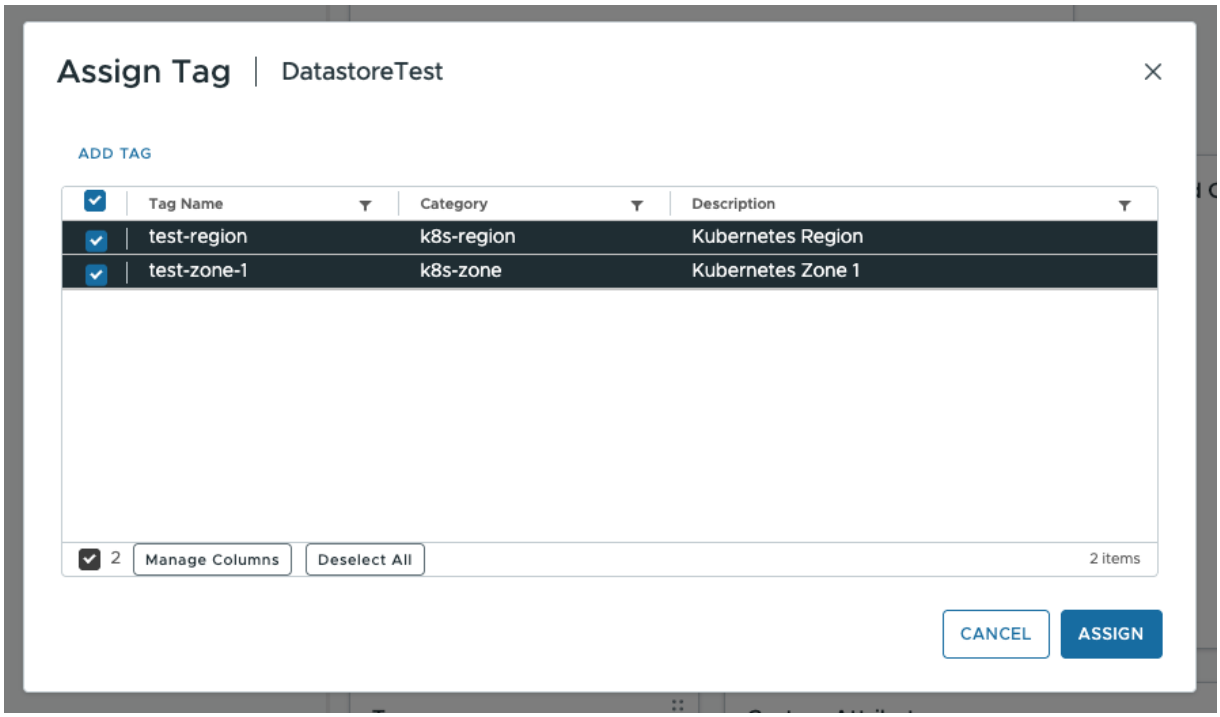
5. In the “Inventory” tab, select the target Datacenter, open the “Summary” panel, then choose “Actions” → “Tags & Custom Attributes” → “Assign Tag” and assign the region tag. Repeat this step for each Cluster that will host nodes, assigning the appropriate zone tags.



Configuring Datastore in vSphere Client

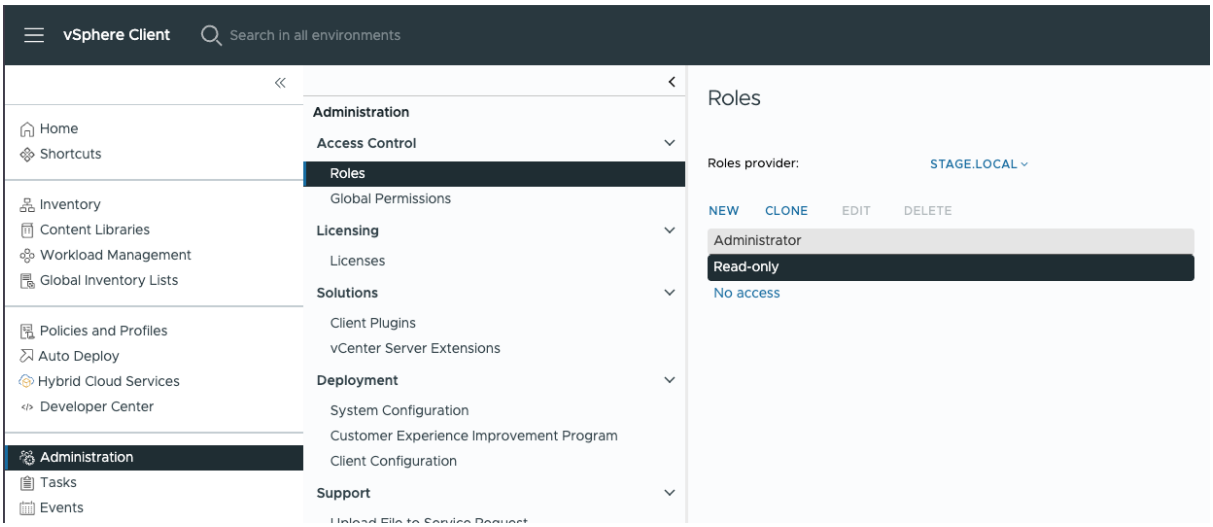
> ⚠ For dynamic provisioning of PersistentVolume, the Datastore must be available on **every** ESXi host in the zone (shared datastore).

In the “Inventory” tab, select the Datastore, open the “Summary” panel, then choose “Actions” → “Tags & Custom Attributes” → “Assign Tag”. Assign the Datastore the same region tag as the corresponding Datacenter, and the same zone tag as the corresponding Cluster.

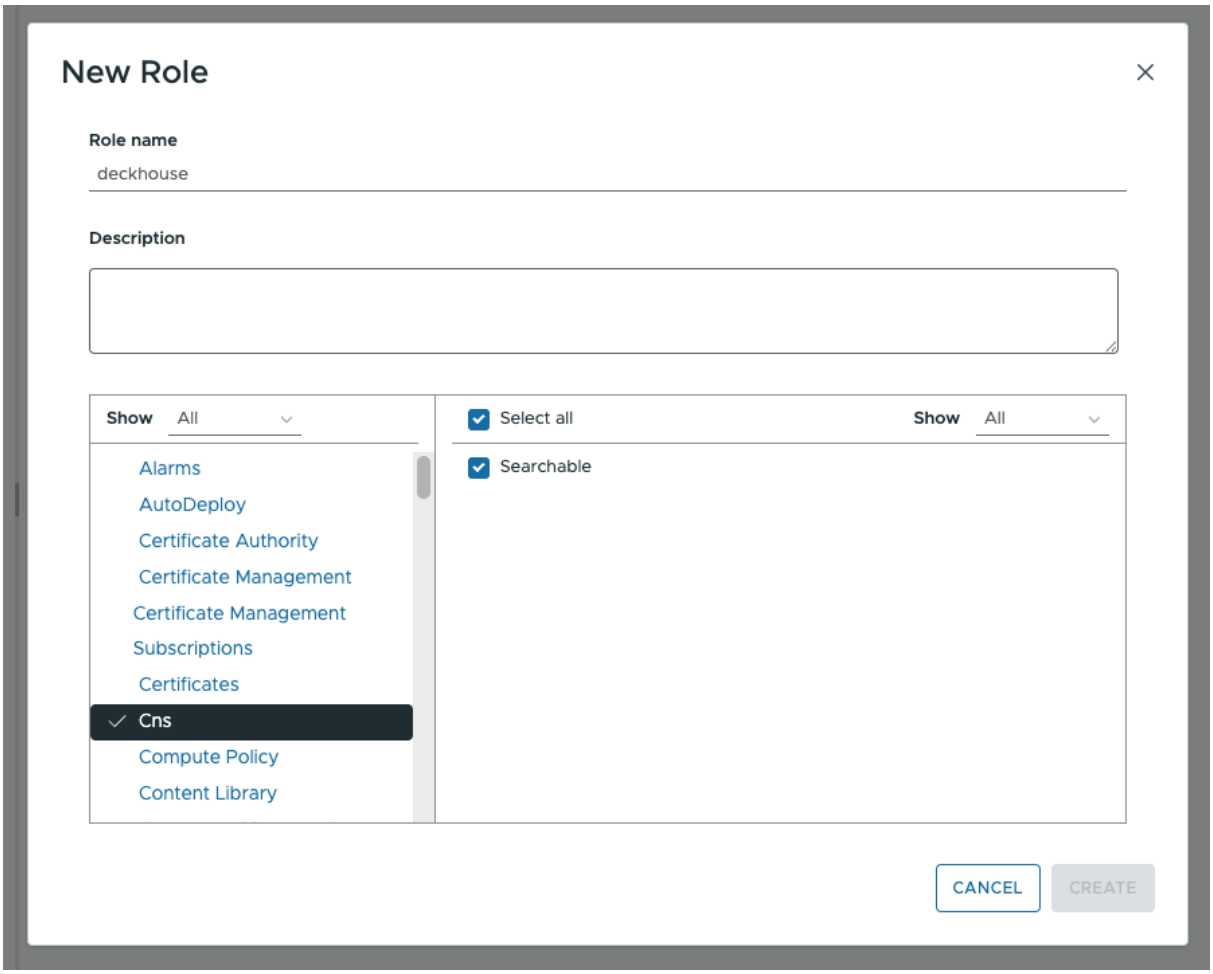


Creating and assigning a role in vSphere Client

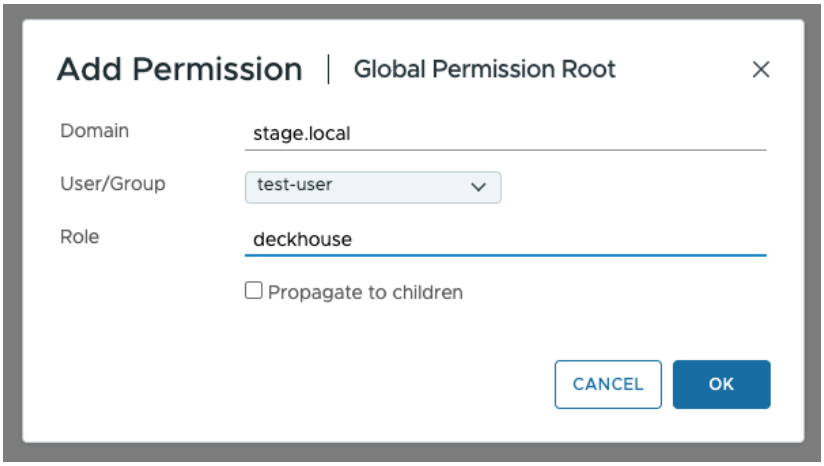
1. Go to “Menu” → “Administration” → “Access Control” → “Roles”.



2. Click “NEW”, enter a role name (for example, `deckhouse`), and add the privileges from the list.



3. Assign the role to the Deckhouse service account: go to “Menu” → “Administration” → “Access Control” → “Global Permissions”, click “ADD”, and select the user and the `deckhouse` role.



Configuration with govc

Installing govc

To continue configuring vSphere, install the govc CLI utility.

After installation, set the environment variables required to connect to `vCenter`.

⚠ Make sure to specify the username together with the domain, for example: `username@domain.local`.

```
export GOVC_URL=example.com
export GOVC_USERNAME=<username>@vsphere.local
export GOVC_PASSWORD=<password>
export GOVC_INSECURE=1
```

Creating tags and tag categories with govc

VMware vSphere does not have built-in concepts of a “region” or a “zone”. In vSphere, a Datacenter acts as a region, and

a Cluster acts as a zone. Tags are used to establish this mapping.

Create tag categories with the following commands:

```
govc tags.category.create -d "Kubernetes Region" k8s-region
govc tags.category.create -d "Kubernetes Zone" k8s-zone
```

Create tags in each category. If you intend to use multiple “zones” (Cluster), create a tag for each one of them:

```
govc tags.create -d "Kubernetes Region" -c k8s-region test-region
govc tags.create -d "Kubernetes Zone Test 1" -c k8s-zone test-zone-1
govc tags.create -d "Kubernetes Zone Test 2" -c k8s-zone test-zone-2
```

Attach the “region” tag to Datacenter:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>
```

Attach “zone” tags to the Cluster objects:

```
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/host/<ClusterName1>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/host/<ClusterName2>
```

Datastore configuration with govc

⚠ For dynamic PersistentVolume provisioning, a Datastore must be available on **each** ESXi host (shared datastore).

Assign the “region” and “zone” tags to the Datastore objects to automatically create a StorageClass in the Kubernetes cluster:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName1>
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/datastore/<DatastoreName1>

govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName2>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/datastore/<DatastoreName2>
```

Creating and assigning a role with govc

i We’ve intentionally skipped User creation since there are many ways to authenticate a user in the vSphere.

The role described below includes the privileges from the list of required privileges section. If you need a more granular Role, please contact your Deckhouse support.

Create a role with the required privileges:

```
govc role.create deckhouse \
    Cns.Searchable \
    Datastore.AllocateSpace Datastore.Browse Datastore.FileManagement \
    Folder.Create Folder.Delete Folder.Move Folder.Rename \
    Global.GlobalTag Global.SystemTag \
    InventoryService.Tagging.AttachTag InventoryService.Tagging.CreateCategory \
    InventoryService.Tagging.CreateTag InventoryService.Tagging.DeleteCategory \
    InventoryService.Tagging.DeleteTag InventoryService.Tagging.EditCategory \
    InventoryService.Tagging.EditTag InventoryService.Tagging.ModifyUsedByForCategory \
    InventoryService.Tagging.ModifyUsedByForTag InventoryService.Tagging.ObjectAttachable \
    Network.Assign \
    Resource.AssignVMToPool Resource.CreatePool Resource.DeletePool Resource.EditPool Resource.RenamePool \
    StorageProfile.View \
    System.Anonymous System.Read System.View \
    VApp.ApplicationConfig VApp.AssignResourcePool VApp.AssignVM VApp.Create VApp.Delete \
    VApp.ExtractOvfEnvironment VApp.Import VApp.InstanceConfig VApp.PowerOff VApp.PowerOn VApp.ResourceConfig \
    VirtualMachine.Config.AddExistingDisk VirtualMachine.Config.AddNewDisk VirtualMachine.Config.AddRemoveDevice \
    VirtualMachine.Config.AdvancedConfig VirtualMachine.Config.Annotation VirtualMachine.Config.CPUCount \
    VirtualMachine.Config.ChangeTracking VirtualMachine.Config.DiskExtend VirtualMachine.Config.DiskLease \
    VirtualMachine.Config.EditDevice VirtualMachine.Config.ManagedBy VirtualMachine.Config.Memory \
    VirtualMachine.Config.QueryUnownedFiles VirtualMachine.Config.RawDevice VirtualMachine.Config.ReloadFromPath \
    VirtualMachine.Config.RemoveDisk VirtualMachine.Config.Rename VirtualMachine.Config.ResetGuestInfo \
    VirtualMachine.Config.Resource VirtualMachine.Config.Settings VirtualMachine.Config.SwapPlacement \
    VirtualMachine.Config.UpgradeVirtualHardware \
    VirtualMachine.GuestOperations.Query \
    VirtualMachine.Interact.AnswerQuestion VirtualMachine.Interact.DeviceConnection \
    VirtualMachine.Interact.GuestControl VirtualMachine.Interact.PowerOff VirtualMachine.Interact.PowerOn \
    VirtualMachine.Interact.Reset VirtualMachine.Interact.SetCDMedia VirtualMachine.Interact.ToolsInstall \
    VirtualMachine.Inventory.Create VirtualMachine.Inventory.CreateFromExisting VirtualMachine.Inventory.Delete \
    VirtualMachine.Inventory.Move \
    VirtualMachine.Provisioning.Clone VirtualMachine.Provisioning.Customize VirtualMachine.Provisioning.DeployTemplate \
    VirtualMachine.Provisioning.GetVmFiles VirtualMachine.Provisioning.PutVmFiles VirtualMachine.Provisioning.ReadCustSpecs
\
    VirtualMachine.State.CreateSnapshot VirtualMachine.State.RemoveSnapshot VirtualMachine.State.RenameSnapshot
```

Assign the role to a user on the vCenter object.

⚠ Make sure to specify the username together with the domain, for example: `username@domain.local`.

```
govc permissions.set -principal <username>@vsphere.local -role deckhouse /
```

i For more detailed permission configuration, refer to the official documentation.

Additional resources

- Module `cloud-provider-vsphere` documentation

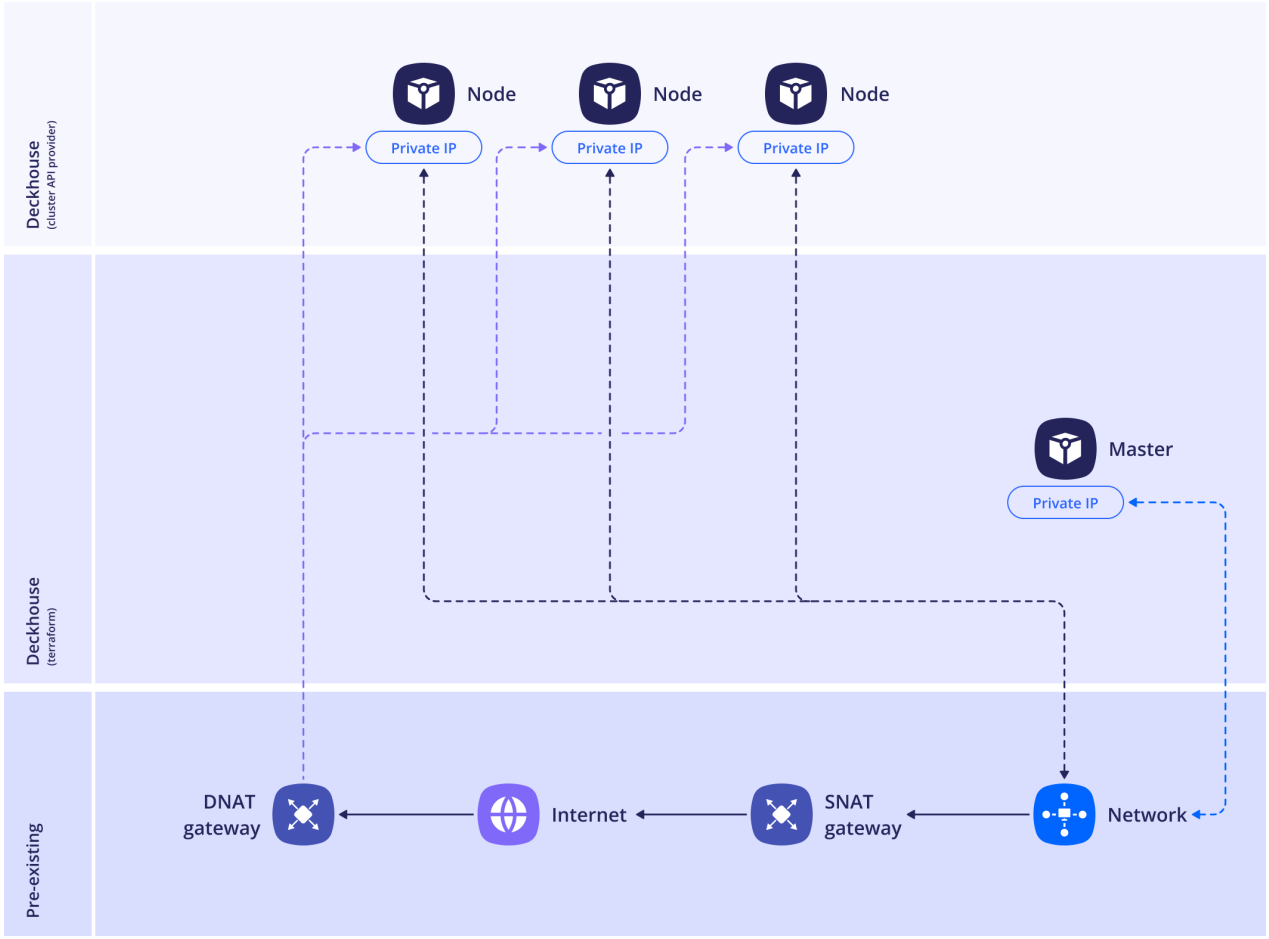
Layouts and configuration in VMware vSphere

Standard

The Standard layout is intended for deploying a cluster within the vSphere infrastructure with full control over resources, networking, and storage.

Key features:

- Uses a vSphere Datacenter as a `region`.
- Uses a vSphere Cluster as a `zone`.
- Supports multiple zones and node placements across zones.
- Supports using different datastores for disks and volumes.
- Supports network connectivity including additional network isolation (for example, MetalLB + BGP).



Example configuration:

```
apiVersion: deckhouse.io/v1
kind: VsphereClusterConfiguration
layout: Standard
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
vmFolderPath: dev
regionTagCategory: k8s-region
zoneTagCategory: k8s-zone
region: X1
internalNetworkCIDR: 192.168.199.0/24
masterNodeGroup:
  replicas: 1
  zones:
    - ru-central1-a
    - ru-central1-b
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: dev/golden_image
    datastore: dev/lun_1
    mainNetwork: net3-k8s
nodeGroups:
  - name: khm
    replicas: 1
    zones:
      - ru-central1-a
    instanceClass:
      numCPUs: 4
      memory: 8192
      template: dev/golden_image
      datastore: dev/lun_1
      mainNetwork: net3-k8s
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
  - ru-central1-a
  - ru-central1-b
```

Required parameters for the VsphereClusterConfiguration resource:

- `region`: Tag assigned to the Datacenter object.
- `zoneTagCategory` and `regionTagCategory`: Tag categories used to identify regions and zones.
- `internalNetworkCIDR`: Subnet for assigning internal IP addresses.
- `vmFolderPath`: Path to the folder where cluster virtual machines will be placed.
- `sshPublicKey`: Public SSH key used to access the nodes.
- `zones`: List of zones available for node placement.

> **i** All nodes placed in different zones must have access to shared datastores with matching zone tags.

List of required privileges

> **i** To create a role and assign it to a user, refer to Configuration in vSphere Client and Configuration with govc sections.

A detailed list of privileges required for Deckhouse Kubernetes Platform to work in vSphere:

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

—	— (assigned by default when creating a role)	`System.Anonymous`
		`System.Read`
		`System.View`
			Basic access to vSphere Inventory objects required for all Deckhouse vSphere integration components.

Cns	Searchable	`Cns.Searchable`	Search and mapping of Container Native Storage objects when the CSI driver works with Kubernetes volumes.

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

Datastore	Allocate space,
	Browse datastore,
	Low level file operations
		`Datastore.AllocateSpace`
		`Datastore.Browse`
		`Datastore.FileManagement`
			Disk provisioning when creating virtual machines and ordering `PersistentVolumes` in the cluster.

Folder	Create folder,
	Delete folder,
	Move folder,
	Rename folder
		`Folder.Create`
		`Folder.Delete`
		`Folder.Move`
		`Folder.Rename`
			Grouping a Deckhouse Kubernetes Platform cluster in a single `Folder` in vSphere Inventory.

Global	Global tag, System tag	`Global.GlobalTag` `Global.SystemTag`	Access to global and system tags used by Deckhouse Kubernetes Platform when working with vSphere objects.

vSphere Tagging	Assign or Unassign vSphere Tag,
	Assign or Unassign vSphere Tag on Object,
	Create vSphere Tag,
	Create vSphere Tag Category,
	Delete vSphere Tag,
	Delete vSphere Tag Category,
	Edit vSphere Tag,
	Edit vSphere Tag Category,
	Modify UsedBy Field for Category,
	Modify UsedBy Field for Tag
		`InventoryService.Tagging.AttachTag`
		`InventoryService.Tagging.ObjectAttachable`
		`InventoryService.Tagging.CreateTag`
		`InventoryService.Tagging.CreateCategory`
		`InventoryService.Tagging.DeleteTag`
		`InventoryService.Tagging.DeleteCategory`
		`InventoryService.Tagging.EditTag`
		`InventoryService.Tagging.EditCategory`
		`InventoryService.Tagging.ModifyUsedByForCategory`
		`InventoryService.Tagging.ModifyUsedByForTag`
			Deckhouse Kubernetes Platform uses tags to identify the `Datacenter`, `Cluster`, and `Datastore` objects available to it, as well as to identify the virtual machines under its control.

Network	Assign network	`Network.Assign`	Connecting networks and port groups to Deckhouse Kubernetes Platform cluster virtual machines.

Resource	Assign virtual machine to resource pool,
	Create resource pool,
	Modify resource pool,
	Remove resource pool,
	Rename resource pool
		`Resource.AssignVMToPool`
		`Resource.CreatePool`
		`Resource.DeletePool`
		`Resource.EditPool`
		`Resource.RenamePool`
			Placement of Deckhouse Kubernetes Platform cluster virtual machines into the target resource pool and management of this pool.

VM Storage Policies (*Profile-driven Storage Privileges* in vSphere 7)	View VM storage policies (*Profile-driven storage view* in vSphere 7)	`StorageProfile.View`	Viewing storage policies used when creating virtual machines and dynamically provisioning volumes in the cluster.

vApp	Add virtual machine,
	Assign resource pool,
	Create,
	Delete,
	Import,
	Power Off,
	Power On,
	View OVF Environment,
	vApp application configuration,
	vApp instance configuration,
	vApp resource configuration
		`VApp.ApplicationConfig`
		`VApp.AssignResourcePool`
		`VApp.AssignVM`
		`VApp.Create`
		`VApp.Delete`
		`VApp.ExtractOvfEnvironment`
		`VApp.Import`
		`VApp.InstanceConfig`
		`VApp.PowerOff`
		`VApp.PowerOn`
		`VApp.ResourceConfig`
			Managing operations related to deployment and configuration of vApp and OVF templates used when creating virtual machines.

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

Virtual Machine > Change Configuration	Add existing disk,
	Add new disk,
	Add or remove device,
	Advanced configuration,
	Set annotation,
	Change CPU count,
	Toggle disk change tracking,
	Extend virtual disk,
	Acquire disk lease,
	Modify device settings,
	Configure managedBy,
	Change Memory,
	Query unowned files,
	Configure Raw device,
	Reload from path,
	Remove disk,
	Rename,
	Reset guest information,
	Change resource,
	Change Settings,
	Change Swapfile placement,
	Upgrade virtual machine compatibility
		`VirtualMachine.Config.AddExistingDisk`
		`VirtualMachine.Config.AddNewDisk`
		`VirtualMachine.Config.AddRemoveDevice`
		`VirtualMachine.Config.AdvancedConfig`
		`VirtualMachine.Config.Annotation`
		`VirtualMachine.Config.CPUCount`
		`VirtualMachine.Config.ChangeTracking`
		`VirtualMachine.Config.DiskExtend`
		`VirtualMachine.Config.DiskLease`
		`VirtualMachine.Config.EditDevice`
		`VirtualMachine.Config.ManagedBy`
		`VirtualMachine.Config.Memory`
		`VirtualMachine.Config.QueryUnownedFiles`
		`VirtualMachine.Config.RawDevice`
		`VirtualMachine.Config.ReloadFromPath`
		`VirtualMachine.Config.RemoveDisk`
		`VirtualMachine.Config.Rename`
		`VirtualMachine.Config.ResetGuestInfo`
		`VirtualMachine.Config.Resource`
		`VirtualMachine.Config.Settings`
		`VirtualMachine.Config.SwapPlacement`
		`VirtualMachine.Config.UpgradeVirtualHardware`
			Managing the lifecycle of Deckhouse Kubernetes Platform cluster virtual machines.

Virtual Machine > Edit Inventory	Create new,
	Create from existing,
	Remove,
	Move
		`VirtualMachine.Inventory.Create`
		`VirtualMachine.Inventory.CreateFromExisting`
		`VirtualMachine.Inventory.Delete`
		`VirtualMachine.Inventory.Move`
			Creating, deleting, and moving Deckhouse Kubernetes Platform cluster virtual machines in vSphere Inventory.

Virtual Machine > Guest Operations	Guest Operation Queries	`VirtualMachine.GuestOperations.Query`	Retrieving information from the guest operating system of virtual machines.

Virtual Machine > Interaction	Answer question,
	Device connection,
	Guest operating system management by VIX API,
	Power Off,
	Power On,
	Reset,
	Configure CD media,
	Install VMware Tools
		`VirtualMachine.Interact.AnswerQuestion`
		`VirtualMachine.Interact.DeviceConnection`
		`VirtualMachine.Interact.GuestControl`
		`VirtualMachine.Interact.PowerOff`
		`VirtualMachine.Interact.PowerOn`
		`VirtualMachine.Interact.Reset`
		`VirtualMachine.Interact.SetCDMedia`
		`VirtualMachine.Interact.ToolsInstall`
			Managing virtual machine power state, device connections, and interaction with the guest operating system.

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

Virtual Machine > Provisioning	Clone virtual machine,
	Customize guest,
	Deploy template,
	Allow virtual machine download,
	Allow virtual machine files upload,
	Read customization specifications
		`VirtualMachine.Provisioning.Clone`
		`VirtualMachine.Provisioning.Customize`
		`VirtualMachine.Provisioning.DeployTemplate`
		`VirtualMachine.Provisioning.GetVmFiles`
		`VirtualMachine.Provisioning.PutVmFiles`
		`VirtualMachine.Provisioning.ReadCustSpecs`
			Cloning virtual machine templates, customizing them, and deploying them when creating Deckhouse Kubernetes Platform cluster nodes.

Virtual Machine > Snapshot Management	Create snapshot,
	Remove Snapshot,
	Rename Snapshot
		`VirtualMachine.State.CreateSnapshot`
		`VirtualMachine.State.RemoveSnapshot`
		`VirtualMachine.State.RenameSnapshot`
			Managing snapshots of virtual machines and volumes in scenarios where this functionality is used by platform components.

Additional resources

- Module `cloud-provider-vsphere` provider configuration

Storage and load balancing in VMware vSphere

Storage

The following storage types are used in VMware vSphere for Kubernetes cluster data:

- **Datastores**: Used to store the root disks of virtual machines;
- **CNS disks (Container Native Storage)**: Used for automatic creation of PersistentVolumes via CSI.

Deckhouse Kubernetes Platform (DKP) automatically creates a StorageClass for each Datastore and DatastoreCluster that is tagged as a `zone`.

You can specify:

- The default StorageClass name (`default`).
- Exclusions via the `exclude` field in a form of a list of names or patterns for StorageClasses that should not be created.

Example configuration using ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  version: 2
  enabled: true
  settings:
    storageClass:
      default: fast-lun102
      exclude:
        - ".*-lun101-.*"
        - slow-lun103
```

Resizing a volume (PVCs)

DKP supports Online Resize PersistentVolume starting with vSphere 7.0U2. However, due to CSI and vSphere API specifics, additional steps are required after resizing a PVC:

1. Run `d8 k cordon <node_name>`.
2. Delete the Pod that uses the PVC.
3. Wait for the resize operation to complete:
   - Ensure the PVC no longer has the `Resizing` condition.
   - It’s safe to ignore the `FileSystemResizePending` status.
4. Run `d8 k uncordon <node_name>`.

Load balancing

Options for organizing incoming traffic load balancing:

1. **Via an external load balancer**. If your infrastructure includes an external load balancer (for example, NSX-T), you can route traffic directly to the cluster’s frontend nodes.

2. **Via MetalLB**. For fault-tolerant load balancing within the cluster, it is recommended that you use MetalLB in BGP mode. In this case:

- Frontend nodes receive two network interfaces.
- A dedicated VLAN is required for BGP traffic.
- The network must provide DHCP and internet access.
- IP addresses and BGP router ASNs must be specified.
- A pool of IP addresses to be announced must be defined.

> **i** Make sure there is connectivity between BGP routers and frontend nodes in the dedicated VLAN.

CSI

The storage subsystem uses CNS disks by default, with support for online resizing.
Legacy mode with FCD disks is also supported. The subsystem behavior is configured via the `compatibilityFlag` parameter.

Important information on PVC size expansion

Due to specifics of the CSI volume-resizer and vSphere API, after increasing a PVC size, you must perform the following steps:

1. On the node hosting the Pod, run `d8 k cordon <node_name>`.
2. Delete the Pod.
3. Ensure the resize operation completed successfully — the PVC **must not** have the `Resizing` condition.

   > **i** The `FileSystemResizePending` condition is not an issue.

4. On the node hosting the Pod, run `d8 k uncordon <node_name>`

Datastore configuration

> **i** You can also tag **Datastore** objects through the **VMware vSphere Client** — follow Datastore configuration in the connection and authorization guide. The steps below use **`govc` only**.

> ⚠ For dynamic `PersistentVolume` provisioning, a `Datastore` must be available on **each** ESXi host (shared datastore).

For PersistentVolume to function correctly, the datastore must be accessible from all ESXi hosts.

Assign tags:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName1>
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/datastore/<DatastoreName1>

govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName2>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/datastore/<DatastoreName2>
```

Additional resources

- Module `cloud-provider-vsphere` parameters

Integration with VMware vSphere services

Deckhouse Kubernetes Platform integrates with VMware vSphere infrastructure and uses VsphereInstanceClass resources to describe the specifications of virtual machines created as part of the Kubernetes cluster.

Key features:

- Provisioning and removal of virtual machines via the vCenter API.
- Node placement across multiple clusters (`zones`) and datacenters (`region`).
- Use of VM templates with `cloud-init` .
- Support for networks with DHCP, static addressing, and additional interfaces.
- Storage management: provisioning root disks and PVCs based on Datastore or CNS disks.
- Support for incoming traffic load balancing:
  - Via external load balancers.
  - Via MetalLB (in BGP mode).

> **i** DKP supports hybrid integration with VMware vSphere. For configuration details, see Hybrid cluster with vSphere section.

vSphere resource management

Removing CloudPermanent nodes in vSphere

Nodes of the `CloudPermanent` type are created based on the node group configuration specified in the `nodeGroups` section of the VsphereClusterConfiguration resource.

The `replicas` parameter defines the required number of virtual machines in the group. After the configuration is changed, run the `dhctl converge` command to start Terraform and bring the state of virtual machines in VMware vSphere in line with the specified number of replicas.

To reduce the number of nodes in a group, decrease the `replicas` value and run `dhctl converge`.

⚠ Do not remove the group definition from the `nodeGroups` section while the value of the `replicas` parameter is greater than zero.

If the group definition is removed before the number of replicas is reduced to zero, the Terraform state may become inconsistent with the state of virtual machines and disks in VMware vSphere. As a result, a subsequent `dhctl converge` run may fail and require manual state recovery.

Reducing the number of nodes

To reduce the number of nodes in a `CloudPermanent` group:

1. Open the vSphere configuration for editing:

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

2. In the `nodeGroups` section, find the required group and decrease the value of the `replicas` parameter to the required number of nodes.

   For example, to reduce the number of nodes in the `worker` group from three to two, set `replicas: 2`:

   ```
   nodeGroups:
   - name: worker
     replicas: 2
     zones:
     - zone-a
     instanceClass:
       numCPUs: 4
       memory: 8192
       template: Templates/ubuntu-24.04
       datastore: datastore-1
       mainNetwork: network-1
   ```

   Save the changes.

3. In the DKP installer container, apply the updated configuration:

   ```
   dhctl converge \
     --ssh-host <MASTER-NODE-IP-ADDRESS> \
     --ssh-user <USERNAME> \
     --ssh-agent-private-keys /tmp/.ssh/<PRIVATE-SSH-KEY-NAME>
   ```

   i Use an installer container of the same edition and version as the cluster.

4. Wait for `dhctl converge` to complete and check the number of nodes in the group:

   ```
   d8 k get nodes -l node.deckhouse.io/group=<GROUP-NAME>
   ```

Completely removing a node group

A `CloudPermanent` group must be removed in two stages. First, reduce the number of replicas to zero and wait for the nodes and virtual machines to be removed. You can then remove the group definition from the vSphere configuration.

To completely remove a `CloudPermanent` group:

1. Open the vSphere configuration for editing:

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

2. In the `nodeGroups` section, find the group to be removed and set `replicas: 0`. Do not remove the group definition at this stage. For example:

```
nodeGroups:
- name: worker
  replicas: 0
  zones:
  - zone-a
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: Templates/ubuntu-24.04
    datastore: datastore-1
    mainNetwork: network-1
```

Save the changes.

3. In the DKP installer container, apply the updated configuration:

```
dhctl converge \
  --ssh-host <MASTER-NODE-IP-ADDRESS> \
  --ssh-user <USERNAME> \
  --ssh-agent-private-keys /tmp/.ssh/<PRIVATE-SSH-KEY-NAME>
```

4. Wait for `dhctl converge` to complete successfully and make sure that no nodes remain in the group:

```
d8 k get nodes -l node.deckhouse.io/group=<GROUP-NAME>
```

The command must not return any nodes from the group being removed.

Also use vSphere Client to make sure that the virtual machines associated with the group have been removed.

5. Open the cloud provider configuration again:

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

6. Remove the group definition from the `nodeGroups` section and save the changes.
7. Apply the configuration again:

```
dhctl converge \
  --ssh-host <MASTER-NODE-IP-ADDRESS> \
  --ssh-user <USERNAME> \
  --ssh-agent-private-keys /tmp/.ssh/<PRIVATE-SSH-KEY-NAME>
```

8. Make sure that the NodeGroup object has been removed:

```
d8 k get nodegroup <GROUP-NAME>
```

The command must return a message stating that the requested resource was not found.

Additional resources

- Module `cloud-provider-vsphere` custom resources
- Module `cloud-provider-vsphere` provider configuration

Connection and authorization in zVirt

Requirements

To ensure proper operation of Deckhouse Kubernetes Platform with the zVirt cloud, the following are required:

- A working zVirt installation with accessible API.
- zVirt version `4.0`-`4.4`.
- A user account with permissions to access the API and manage virtual machines.
- A storage domain with an uploaded cloud OS image.
- A prepared virtual machine template based on the cloud image.

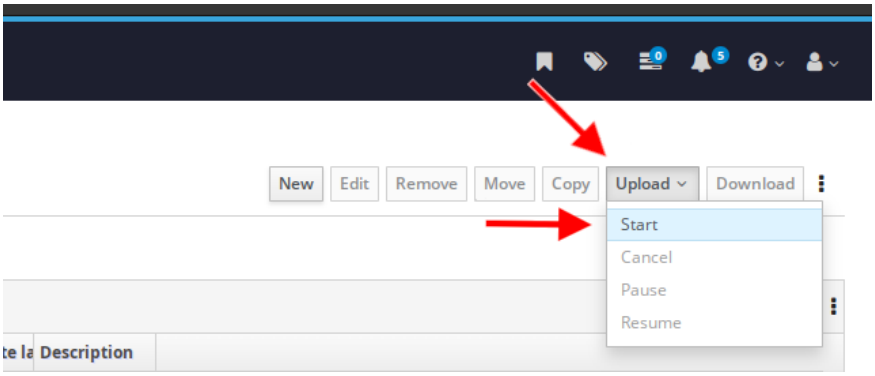
> **i** The zVirt cloud provider is in experimental status. API and behavior may change in future versions.

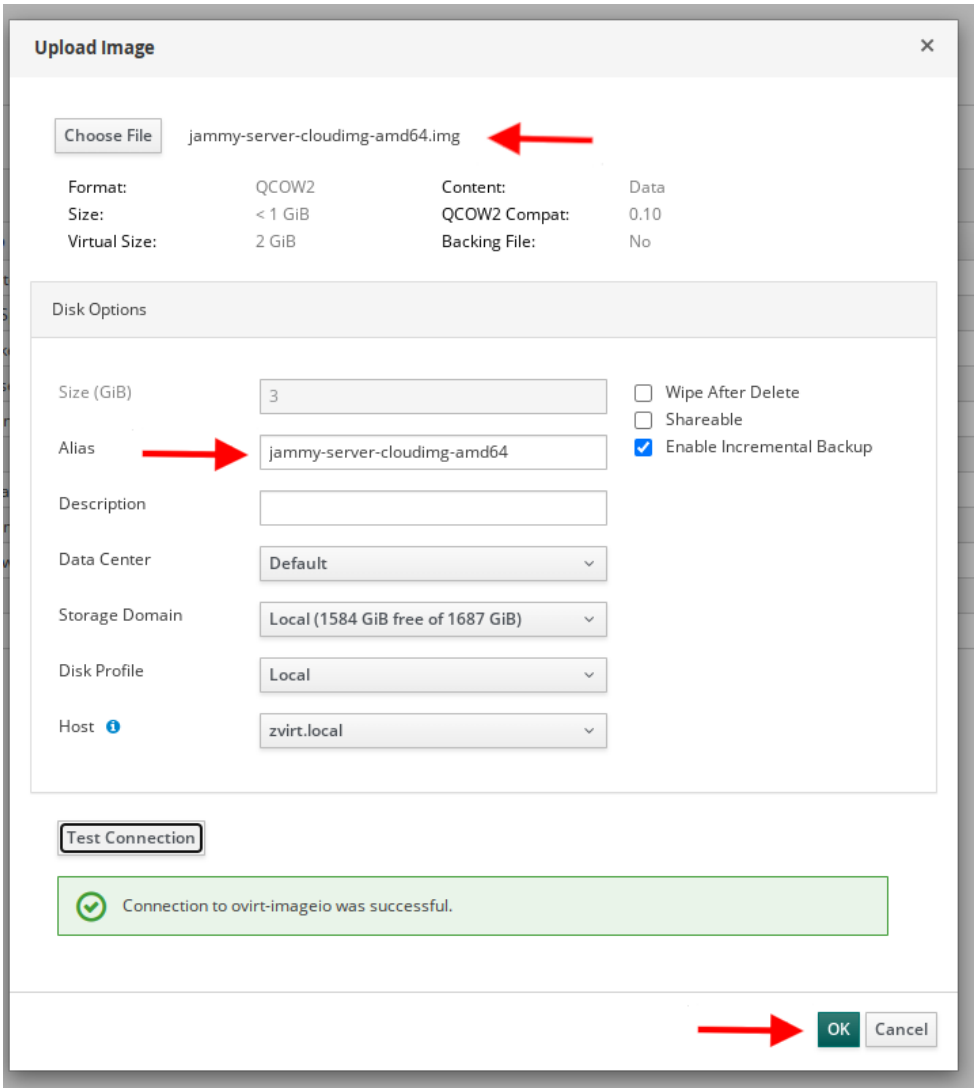
> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Preparing a cloud image

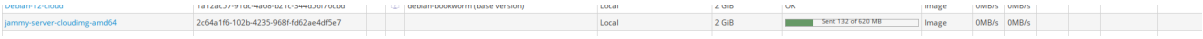
1. Go to **zVirt → Storage → Disks**.

2. Upload a cloud OS image (for example, `.qcow2` or `.img`) to the storage.



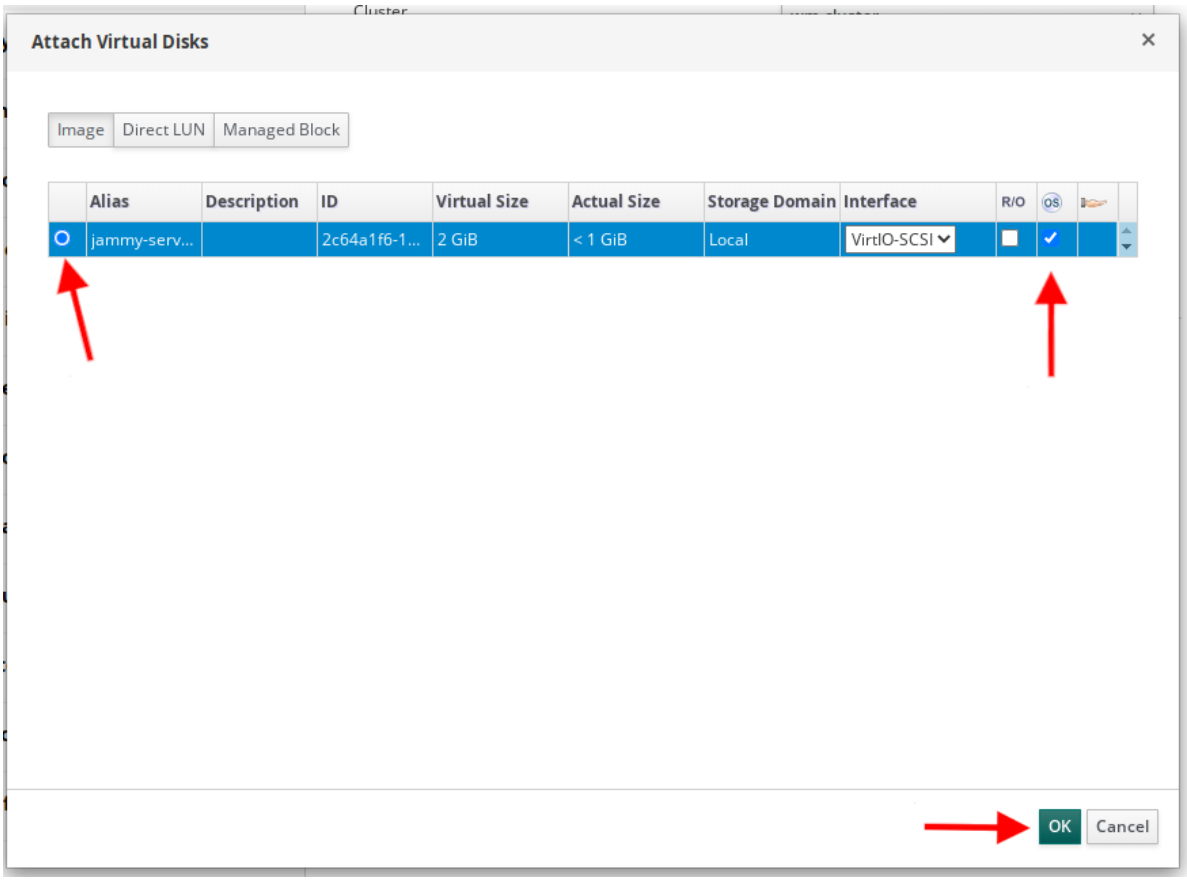


3. Wait for the upload to complete. The disk status should display `OK`.

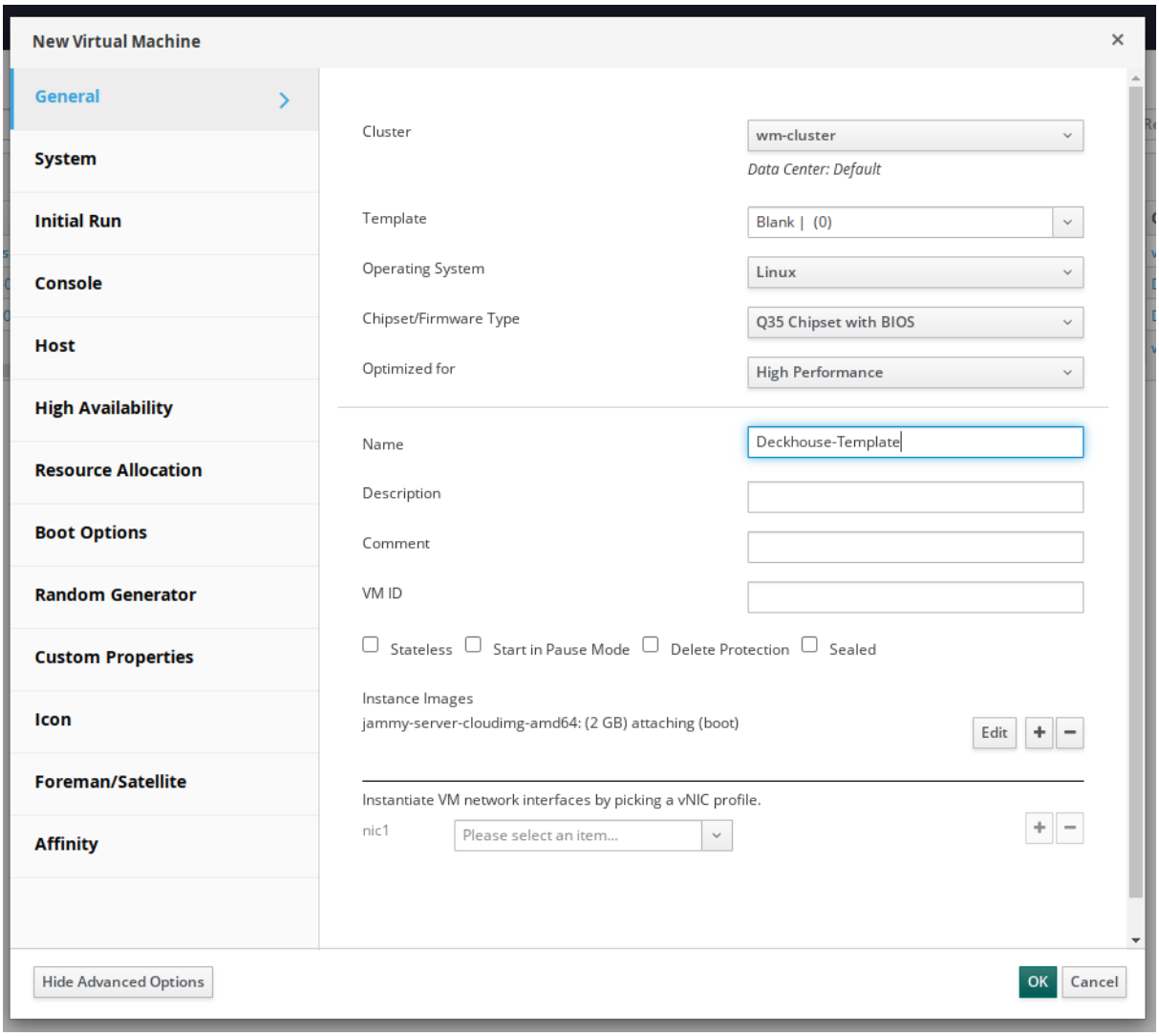


Preparing a virtual machine template

1. Open **Resources → Virtual Machines** and create a new VM.
2. Set the following parameters:
   - **Template**: `Blank`.
   - **Operating system**: Matches the uploaded cloud image.
   - **Workload profile**: High performance.
   - **Disks**: Attach the uploaded cloud image and set it as bootable.

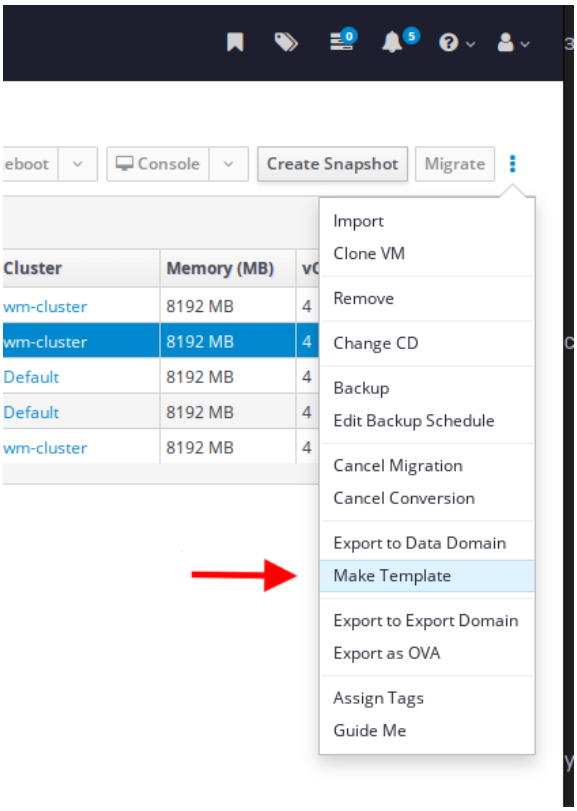


3. Do not add any network interfaces. They will be created automatically when Deckhouse provisions a node.

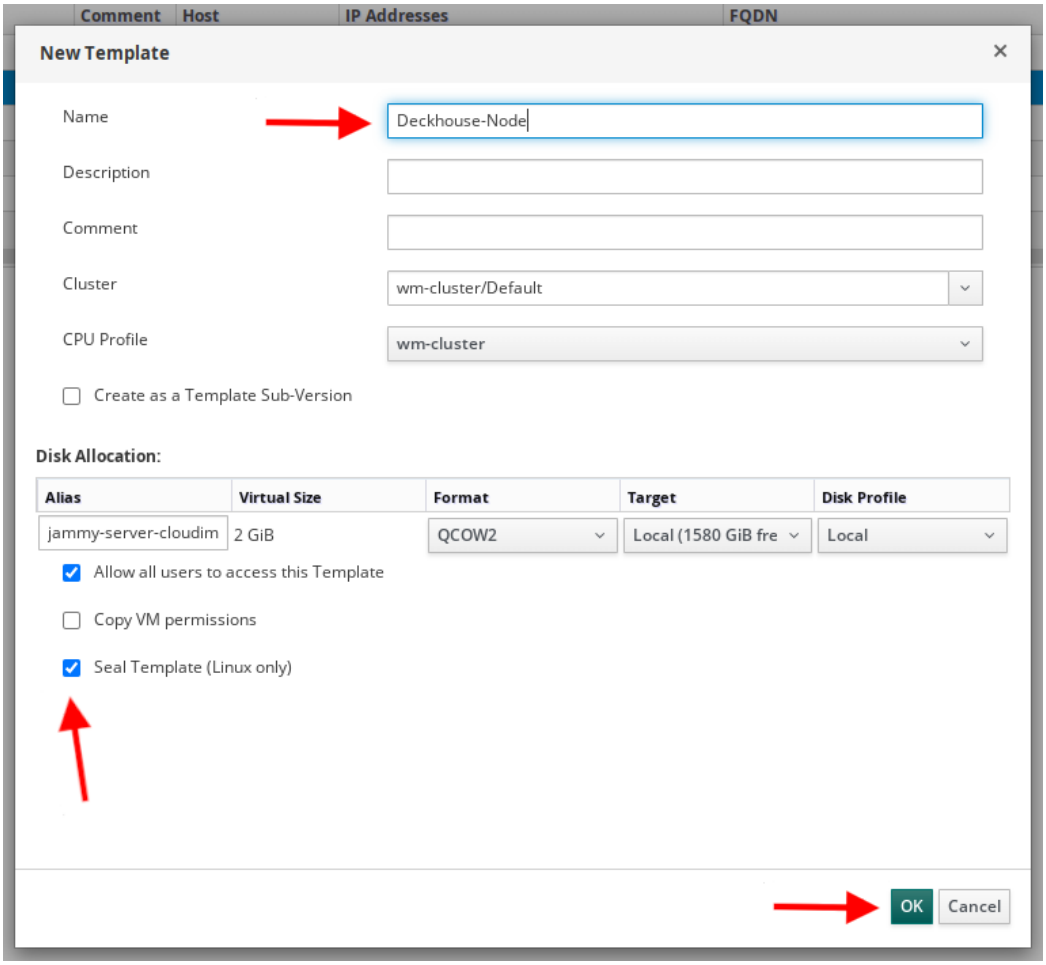


4. Save and create the virtual machine.

5. After the VM is created, open its details and create a template based on it.



6. Specify the template name and enable the **Seal Template** option.



7. Ensure the template has been successfully created in **Resources → Templates**.

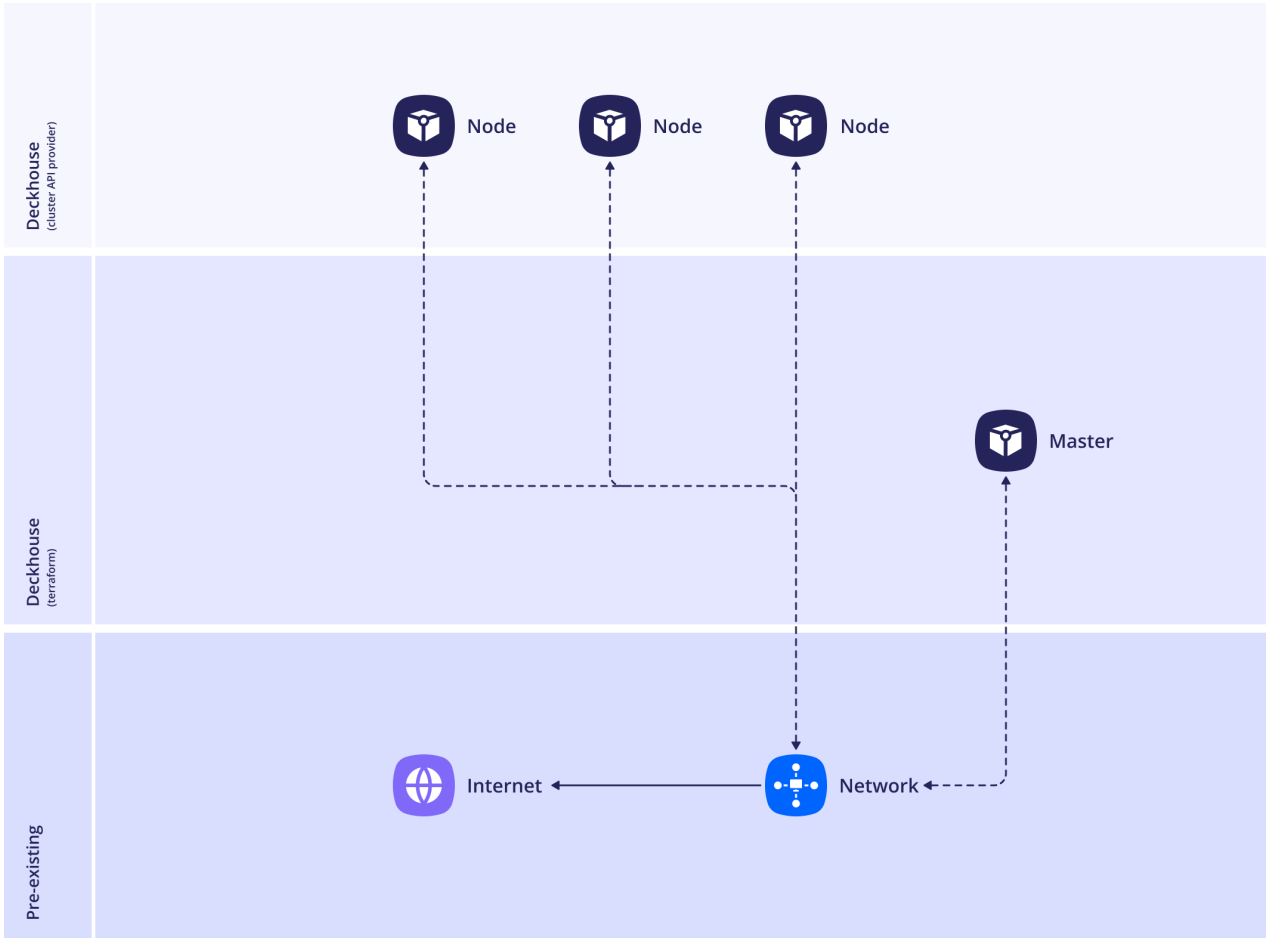
Layouts and configuration in zVirt

Standard

The Standard layout is used to integrate Deckhouse Kubernetes Platform with a zVirt virtual infrastructure. This layout assumes that all nodes are deployed within a single zVirt cluster with centralized management of templates, storage, and networking.

Key features:

- Use of a single zVirt cluster (`clusterID`).
- A storage domain accessible to all hosts in the cluster.
- A virtual machine template created from a cloud image.
- Assignment of a vNIC network profile when provisioning VMs.
- Full automation of node provisioning and removal via the zVirt API.



Configuration example:

```
apiVersion: deckhouse.io/v1
kind: ZvirtClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
clusterID: "b46372e7-0d52-40c7-9bbf-fda31e187088"
provider:
  server: "<SERVER>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    rootDiskSizeGb: 40
    template: ALT-p10
    vnicProfileID: "49bb4594-0cd4-4eb7-8288-8594eafd5a86"
    storageDomainID: "c4bf82a5-b803-40c3-9f6c-b9398378f424"
```

Required parameters for the ZvirtClusterConfiguration resource:

- `clusterID`: UUID of the zVirt cluster where virtual machines will be deployed.
- `sshPublicKey`: Public SSH key used to access the nodes.
- `template`: Name of the prepared VM template.
- `vnicProfileID`: UUID of the vNIC network profile.
- `storageDomainID`: UUID of the storage domain where disks will be placed.

> **i** UUID values (`clusterID`, `vnicProfileID`, and `storageDomainID`) can be obtained via the zVirt API or administrator interface.

Additional resources

- Module `cloud-provider-zvirt` provider configuration

Storage and load balancing in zVirt

Storage

In a cluster hosted on a zVirt infrastructure, Storage Domains are used within the scope of the specified `clusterID`. All virtual machine disks are created within the designated storage domain.

Requirements

- The `storageDomainID` specified in the configuration must be available to the `clusterID` defined in the ZvirtClusterConfiguration resource.
- Disks are created based on the specified `template` and placed in the storage domain.
- For PersistentVolume provisioning, VM root disks are used. Separate PVCs are not yet supported in zVirt.

Configuration

Example fragment of ZvirtClusterConfiguration with storage domain reference:

```
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    rootDiskSizeGb: 40
    template: ALT-p10
    vnicProfileID: "49bb4594-0cd4-4eb7-8288-8594eafd5a86"
    storageDomainID: "c4bf82a5-b803-40c3-9f6c-b9398378f424"
```

> **i** Use unique identifiers (UUIDs) to specify the template and storage domain. You can retrieve them through the zVirt API or administrator interface.

Load balancing

The zVirt platform does not provide a built-in load balancer. To handle incoming traffic, the following approaches are recommended:

1. Use an external load balancer. If your infrastructure already has a hardware or software load balancer, configure port forwarding (for example, `80/443`) to the cluster's frontend nodes.
2. Use MetalLB. For fault-tolerant load balancing within the cluster, you can use MetalLB in Layer 2 (L2) mode.

Recommendations:

- Allocate a dedicated L2 network with DHCP and internet access.
- Define an IP address pool that MetalLB can use for announcements.
- Ensure this network is accessible from the cluster's frontend nodes.
- Leave the network interface configuration empty in the VirtualMachine Template. Deckhouse Kubernetes Platform will automatically attach them during VM creation.

> **i** Support for MetalLB in BGP mode is not guaranteed in zVirt and depends on your network infrastructure.

Additional resources

- Module `cloud-provider-zvirt` provider configuration

Integration with zVirt services

> **i** Integration with zVirt is in experimental status. Interfaces and functionality may change in the future.

Deckhouse Kubernetes Platform supports integration with zVirt infrastructure, enabling the provisioning, management, and removal of virtual machines using definitions in the ZvirtInstanceClass resource.

Key features

- Provisioning of virtual machines in zVirt during NodeGroup creation or scaling.
- Use of preconfigured virtual machine templates.

- Automatic configuration of network interfaces and cluster connection.
- Support for both dynamic and static IP address allocation.
- Disk placement in a specified storage domain.
- Support for cloud images with `cloud-init` for proper node initialization.

Additional resources

- Module `cloud-provider-zvirt` custom resources

Hybrid clusters

A hybrid cluster is a static Deckhouse Kubernetes Platform (DKP) cluster extended with nodes placed in the infrastructure of a supported provider, such as Yandex Cloud, VMware Cloud Director or VMware vSphere.

In this scenario, the control plane and the initial cluster nodes remain part of the static DKP cluster, while additional nodes are added through the corresponding provider module. These nodes can be created automatically through the provider API or connected manually if the virtual machines were created in advance.

This approach allows you to expand an existing static cluster without creating a separate Kubernetes cluster: increase compute capacity, place some workloads in another infrastructure, or gradually migrate services there. Applications still use a single Kubernetes control plane: the same API, shared resources, and unified mechanisms for scheduling, monitoring, updates, and operations.

Hybrid integration is performed on top of an already deployed static DKP cluster (`clusterType: Static`). Then the corresponding cloud provider module is enabled, allowing DKP to obtain information about the external infrastructure and create or connect additional nodes.

Depending on the provider, nodes can be added automatically through the provider API or connected manually by running a bootstrap script on pre-created virtual machines.

The general principles of hybrid integration are described on this page. The procedure for preparing the infrastructure and adding nodes depends on the provider being used and is described in separate guides:

- Yandex Cloud;
- VMware Cloud Director;
- VMware vSphere.

NodeGroup types

In a hybrid scenario, DKP uses the following NodeGroup types:

- `CloudEphemeral` — nodes that DKP creates and deletes automatically through the configured cloud provider API.
- `CloudStatic` — nodes that are created manually by the user or by external tools in the same cloud infrastructure for which cloud provider integration is configured. CSI runs on such nodes, and the Node object is managed by cloud-controller-manager: it is automatically enriched with zone and region information based on provider data.

Ways to add nodes

Nodes from the provider infrastructure can be connected in the following ways:

- **Automatic creation of cloud nodes**. DKP creates virtual machines through the provider API. VM parameters are described by the `*InstanceClass` resource, while the required number of nodes and placement zones are defined by the NodeGroup resource with the `CloudEphemeral` type.
- **Connecting manually created cloud nodes using a bootstrap script**. A virtual machine is created in advance by the user in the provider infrastructure, after which a DKP bootstrap script is run on it to connect the node to the cluster. This scenario uses a NodeGroup with the `CloudStatic` type. Such a node is managed by the corresponding provider's cloud-controller-manager.

This section describes the general network requirements for hybrid clusters. Provider-specific requirements, including network parameters, virtual machine templates, credentials, placement parameters, and ways to add nodes, are described in separate guides for Yandex Cloud, VCD, and vSphere.

General network requirements

Two-way network connectivity sufficient for DKP and Kubernetes components must be configured between the nodes of the initial static cluster and the connected nodes placed in the provider infrastructure.

Connectivity must provide:

- access from connected nodes to the Kubernetes API of the initial cluster;
- access from connected nodes to DNS servers;
- access from connected nodes to the container image registry and other external services required to download images and packages;
- access from the control plane and system components to the connected nodes for kubelet, networking components, monitoring, and Kubernetes service operations;
- access from DKP components that interact with the provider infrastructure to the corresponding provider API.

The full list of required connections is provided in the Network interaction section, and access restriction recommendations are provided in the Configuring network policies section.

Additionally, it is recommended to check:

- routing between the networks of the initial static cluster and the connected nodes in both directions;
- consistent MTU values across the entire network path, especially when tunnels are used;
- traffic encapsulation parameters when using Cilium, including `tunnelMode`, if traffic filtering is applied between sites.

Specific requirements for networks, subnets, virtual machine templates, credentials, and additional parameters depend on

the infrastructure provider being used and are described in the “Prerequisites” section for the corresponding provider.

Additional resources

- Global custom resources
- Module `cni-cilium` parameters
- Module `node-manager` custom resources

Hybrid cluster with Yandex Cloud

The following describes the process of adding nodes from Yandex Cloud to an existing static Deckhouse Kubernetes Platform cluster.

Integration with Yandex Cloud uses the `cloud-provider-yandex` module. It provides interaction between DKP and the Yandex Cloud API, retrieval of information about cloud infrastructure, creation of virtual machines, work with network parameters, and connection of nodes to an existing cluster.

This section describes two ways to add nodes:

- **Automatic node creation in Yandex Cloud**. DKP creates virtual machines through the Yandex Cloud API. VM parameters are defined by the YandexInstanceClass resource, and the required number of nodes and placement zones are defined by the NodeGroup resource with the `CloudEphemeral` type.
- **Connecting manually created nodes through a bootstrap script**. A virtual machine is created by the user in advance and connected to the cluster using the DKP bootstrap script. This scenario uses NodeGroup with the `CloudStatic` type.

Prerequisites

Before you begin, make sure that the following conditions are met:

- The cluster was created with the `clusterType: Static` parameter.
- Network connectivity is configured between the network of static nodes and the Yandex Cloud VPC. Yandex Cloud nodes added to the cluster have access to the Kubernetes API, DNS, and the required addresses according to the Network interaction and Network policy configuration sections. When using Cilium with pod traffic tunneling, the `tunnelMode` mode is selected according to the network connectivity between sites.
- The requirements from the Connection and authorization in Yandex Cloud section are met:
  - A service account is prepared.
  - A folder where resources will be created is selected.
  - The required roles and access to the VPC being used are configured.

Adding automatically created nodes

To run the preparation commands, you need the Yandex Cloud CLI (`yc`). You can use it on the administrator’s workstation. The `yc` CLI is not required on the cluster master node: only the prepared manifests need to be applied in the cluster.

1. Get the identifiers of the cloud and folder where nodes will be created:

   ```
   yc resource-manager cloud list
   yc resource-manager folder list
   ```

2. Specify the obtained identifiers in variables:

   ```
   export CLOUD_ID="<CLOUD_ID>"
   export FOLDER_ID="<FOLDER_ID>"
   ```

   Where:

   - `CLOUD ID` — Yandex Cloud cloud ID;
   - `FOLDER_ID` — ID of the folder where resources will be created.

3. Get the identifiers of the network, subnet, and zone where nodes will be created:

   ```
   yc vpc network list --folder-id "$FOLDER_ID"
   yc vpc subnet list --folder-id "$FOLDER_ID"
   yc compute zone list
   ```

4. Specify the obtained values in variables:

   ```
   export NETWORK_ID="<NETWORK_ID>"
   export SUBNET_ID="<SUBNET_ID>"
   export ZONE="<ZONE>"
   ```

   Where:

   - `NETWORK ID` — VPC network ID;
   - `SUBNET_ID` — ID of the subnet where nodes will be created;
   - `ZONE` — availability zone that corresponds to the selected subnet, for example `ru-central1-a`.

   For details, see Connecting and authorizing in Yandex Cloud.

5. Create a service account in the required Yandex Cloud folder and assign permissions to it:

```
yc iam service-account create \
  --name dkp-hybrid \
  --folder-id "$FOLDER_ID"

export SA_ID="$(yc iam service-account get \
  --name dkp-hybrid \
  --folder-id "$FOLDER_ID" \
  --format json | jq -r .id)"

yc resource-manager folder add-access-binding "$FOLDER_ID" \
  --role editor \
  --subject "serviceAccount:${SA_ID}"

yc resource-manager folder add-access-binding "$FOLDER_ID" \
  --role vpc.admin \
  --subject "serviceAccount:${SA_ID}"
```

The `editor` role is required to create and manage cloud resources, and `vpc.admin` is required to work with VPC network resources.

6. Create a service account key and save it to a JSON file:

   ```
   yc iam key create \
     --service-account-id "$SA_ID" \
     --output dkp-hybrid-sa-key.json
   ```

   Save the service account JSON key to the `SERVICE_ACCOUNT_JSON` environment variable in single-line format:

   ```
   export SERVICE_ACCOUNT_JSON="$(jq -c . dkp-hybrid-sa-key.json)"
   ```

7. Save the administrator's public SSH key to the `SSH_PUBLIC_KEY` environment variable:

   ```
   export SSH_PUBLIC_KEY="$(cat ~/.ssh/id_rsa.pub)"
   ```

   If you use another key, specify its path instead of `~/.ssh/id_rsa.pub`.

   ⚠ The `SSH_PUBLIC_KEY` variable must contain the administrator's public SSH key that will be used to access the nodes being created. Do not use the public key from the service account JSON file.

8. Get the ID of the operating system image that will be used to create virtual machines and save it to the `IMAGE_ID` environment variable:

   ```
   export IMAGE_ID="$(yc compute image get-latest-from-family ubuntu-2404-lts \
     --folder-id standard-images \
     --format json | jq -r .id)"
   ```

   ⚠ The `IMAGE_ID` variable must contain the OS image ID in Yandex Cloud. Do not use an existing virtual machine ID or a service account key ID.

9. Specify the CIDR of the network where Yandex Cloud nodes will be placed:

   ```
   export NODE_NETWORK_CIDR="<NODE_NETWORK_CIDR>"
   ```

   `NODE_NETWORK_CIDR` is the CIDR that includes the internal IP addresses of Yandex Cloud nodes. For a single zone, it usually matches the CIDR of the selected subnet. For example, if nodes are created in the `10.128.0.0/24` subnet, specify `10.128.0.0/24`. You can get the subnet CIDR with the following command:

   ```
   yc vpc subnet list --folder-id "$FOLDER_ID"
   ```

10. Create a provider configuration file. For example, `cloud-provider-cluster-configuration.yaml`:

```
cat > cloud-provider-cluster-configuration.yaml <<EOF
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: WithoutNAT
masterNodeGroup:
  replicas: 1
  instanceClass:
    cores: 4
    memory: 8192
    imageID: ${IMAGE_ID}
    diskSizeGB: 100
    platform: standard-v3
    externalIPAddresses:
      - "Auto"
nodeNetworkCIDR: ${NODE_NETWORK_CIDR}
existingNetworkID: empty
provider:
  cloudID: ${CLOUD_ID}
  folderID: ${FOLDER_ID}
  serviceAccountJSON: '${SERVICE_ACCOUNT_JSON}'
sshPublicKey: '${SSH_PUBLIC_KEY}'
EOF
```

The manifest automatically uses the values of the environment variables set in the previous steps: `CLOUD_ID`, `FOLDER_ID`, `IMAGE_ID`, `NODE_NETWORK_CIDR`, `SERVICE_ACCOUNT_JSON`, and `SSH_PUBLIC_KEY`.

> **i** In a hybrid scenario where the control plane is already deployed as a static cluster, the `masterNodeGroup` section does not create master nodes in Yandex Cloud, but remains part of the provider configuration.

11. Create a file with Yandex Cloud discovery data. For example, `cloud-provider-discovery-data.json`:

```
cat > cloud-provider-discovery-data.json <<EOF
{
  "apiVersion": "deckhouse.io/v1",
  "defaultLbTargetGroupNetworkId": "empty",
  "internalNetworkIDs": [
    "${NETWORK_ID}"
  ],
  "kind": "YandexCloudDiscoveryData",
  "monitoringAPIKey": "",
  "region": "ru-central1",
  "routeTableID": "empty",
  "shouldAssignPublicIPAddress": false,
  "zoneToSubnetIdMap": {
    "${ZONE}": "${SUBNET_ID}"
  },
  "zones": [
    "${ZONE}"
  ]
}
EOF
```

The file automatically uses the values of the environment variables set in the previous steps: `NETWORK_ID`, `SUBNET_ID`, and `ZONE`.

The `shouldAssignPublicIPAddress` parameter controls whether public IP addresses are assigned to the nodes being created. In this example, it is set to `false`, so the created nodes will receive only internal IP addresses.

> **⚠** If `shouldAssignPublicIPAddress` is set to `false`, the created nodes must have access to the image registry and external services through a NAT Gateway, NAT instance, proxy, or another egress mechanism. For zones where no subnets are available, the `empty` value is allowed.

12. Encode the `cloud-provider-cluster-configuration.yaml` and `cloud-provider-discovery-data.json` files in Base64:

```
export CLUSTER_CONFIGURATION_B64="$(base64 -w0 cloud-provider-cluster-configuration.yaml)"
export DISCOVERY_DATA_B64="$(base64 -w0 cloud-provider-discovery-data.json)"
```

13. Create a manifest with the `d8-provider-cluster-configuration` secret and ModuleConfig to enable and configure the `cloud-provider-yandex` module:

```
cat > yandex-provider-secret-and-mc.yaml <<EOF
apiVersion: v1
kind: Secret
metadata:
  labels:
    name: d8-provider-cluster-configuration
  name: d8-provider-cluster-configuration
  namespace: kube-system
type: Opaque
data:
  cloud-provider-cluster-configuration.yaml: ${CLUSTER_CONFIGURATION_B64}
  cloud-provider-discovery-data.json: ${DISCOVERY_DATA_B64}
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 1
  enabled: true
  settings:
    storageClass:
      default: network-ssd
EOF
```

14. Copy the `yandex-provider-secret-and-mc.yaml` file to the cluster master node. Apply the manifest:

```
d8 k apply -f yandex-provider-secret-and-mc.yaml
```

15. Wait until the `cloud-provider-yandex` module is enabled and the YandexInstanceClass resource appears:

```
d8 k get module cloud-provider-dvp -o wide
d8 k get crd yandexinstanceclasses.deckhouse.io
```

16. Create a file with YandexInstanceClass and NodeGroup manifests. For example, `yandex-instanceclass-nodegroup.yaml`:

```
apiVersion: deckhouse.io/v1
kind: YandexInstanceClass
metadata:
  name: yc-worker
spec:
  cores: 4
  memory: 8192
  diskSizeGB: 50
  diskType: network-ssd
  mainSubnet: <SUBNET_ID>
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: yc-worker
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: YandexInstanceClass
      name: yc-worker
    minPerZone: 1
    maxPerZone: 1
    zones:
      - ru-central1-a
```

Where:

- YandexInstanceClass describes the parameters of the virtual machine that will be created in Yandex Cloud;
- `mainSubnet` — ID of the subnet from which the created nodes must have access to the static cluster nodes;
- NodeGroup describes the node group that DKP must maintain in the cluster;
- `nodeType: CloudEphemeral` means that nodes will be created automatically through the cloud provider;
- `cloudInstances.zones` must contain zones from the `zones` list in `cloud-provider-discovery-data.json`.

17. Apply the manifest:

```
d8 k apply -f yandex-instanceclass-nodegroup.yaml
```

After the manifest is applied, DKP will start creating a virtual machine in Yandex Cloud through machine-controller-manager.

18. Check that the node appears in the cluster:

    ```
    d8 k get nodes -o wide
    ```

    Example expected output:

    ```
    NAME                                STATUS   ROLES                  AGE   VERSION    INTERNAL-
    IP
    static-master-0                     Ready    control-plane,master   1h    v1.33.10
    10.128.0.15
    yc-worker-f3564dca-7fc59-s2w5d      Ready    yc-worker              10m   v1.33.10
    10.128.0.21
    ```

19. To diagnose the state and troubleshoot possible issues, check the machine-controller-manager logs:

    ```
    d8 k -n d8-cloud-instance-manager get machinedeployments.machine.sapcloud.io -o wide
    d8 k -n d8-cloud-instance-manager get machinesets.machine.sapcloud.io -o wide
    d8 k -n d8-cloud-instance-manager get machines.machine.sapcloud.io -o wide
    d8 k -n d8-cloud-instance-manager logs deploy/machine-controller-manager --tail=200
    ```

Adding manually created nodes through a bootstrap script

Before you begin, make sure that the following conditions are met:

- The `cloud-provider-yandex` module is enabled:

  ```
  d8 k get module cloud-provider-yandex -o wide
  ```

- The `cloud-provider-yandex` module components are in the `Running` state:

  ```
  d8 k -n d8-cloud-provider-yandex get pods -o wide
  ```

- A virtual machine that will be connected to the cluster has been created in Yandex Cloud.
- The virtual machine is connected to the Yandex Cloud network and subnet used for hybrid integration with the cluster.
- The virtual machine has a network interface in the Yandex Cloud VPC network and subnet used for hybrid integration with the cluster. The IP address of this interface must belong to the CIDR specified in `nodeNetworkCIDR` and be reachable from the static cluster nodes.
- The virtual machine name in Yandex Cloud matches the hostname inside the operating system.
- One of the package managers (`apt`/`apt-get`, `yum`, or `rpm`) for a supported OS is installed on the virtual machine.

1. Check the virtual machine metadata in Yandex Cloud.

   The VM metadata must have `cloud-init` configured with the user that will be used for SSH connection.

   Example metadata:

   ```
   #cloud-config
   datasource:
     Ec2:
       strict_id: false
   ssh_pwauth: no
   users:
     - name: <USER>
       sudo: ALL=(ALL) NOPASSWD:ALL
       shell: /bin/bash
       ssh_authorized_keys:
         - <SSH_PUBLIC_KEY>
   ```

   Where:

   - `<USER>`: Username for SSH access to the virtual machine.
   - `<SSH_PUBLIC_KEY>`: Administrator’s public SSH key.

2. On the master node, create a file with the NodeGroup manifest and specify the node group name. In this example and the following steps, the `yc-manual` name is used. For example, `yandex-manual-nodegroup.yaml`:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: yc-manual
spec:
  nodeType: CloudStatic
EOF
```

3. Make sure that the NodeGroup has been created and synchronized:

   ```
   d8 k get nodegroup yc-manual
   d8 k describe nodegroup yc-manual
   ```

   Example expected output:

   ```
   NAME        TYPE          READY   NODES   UPTODATE   INSTANCES   DESIRED   MIN   MAX   STANDBY
   STATUS   AGE   SYNCED
   yc-manual   CloudStatic   0       0       0
   1m    True
   ```

4. On the master node, get the bootstrap script for the created NodeGroup:

   ```
   NODE_GROUP=yc-manual

   d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} \
     -o jsonpath='{.data.bootstrap\.sh}' > ${NODE_GROUP}-bootstrap.b64
   ```

5. Copy the bootstrap script to the VM being connected. If SSH access to the VM is available from the master node, run the following on the master node:

   ```
   scp ${NODE_GROUP}-bootstrap.b64 <USER>@<NODE_PUBLIC_OR_INTERNAL_IP>:/tmp/bootstrap.b64
   ```

   > **i** To copy and run the bootstrap script, use the user specified in the VM metadata.

   If SSH access to the VM is available only from the administrator’s workstation, first copy the file from the master node to the workstation, and then from the workstation to the VM:

   ```
   scp <MASTER_USER>@<MASTER_IP>:/root/${NODE_GROUP}-bootstrap.b64 ./bootstrap.b64
   scp ./bootstrap.b64 <USER>@<NODE_PUBLIC_OR_INTERNAL_IP>:/tmp/bootstrap.b64
   ```

   Where:

   - `<MASTER_USER>`: User for SSH access to the master node.
   - `<MASTER_IP>`: IP address of the master node.
   - `<USER>`: User on the VM being connected.
   - `<NODE_PUBLIC_OR_INTERNAL_IP>`: Public or internal IP address of the VM being connected.

6. On the VM being connected, decode the bootstrap script, set permissions, and run it:

   ```
   base64 -d /tmp/bootstrap.b64 > /tmp/bootstrap.sh
   chmod +x /tmp/bootstrap.sh

   sudo bash /tmp/bootstrap.sh
   ```

   After the bootstrap script is started, it will install the required components, configure the container runtime and kubelet, and connect the node to the cluster.

7. On the master node, check that the new node has appeared:

   ```
   d8 k get nodes -o wide
   ```

   Example expected output:

   ```
   NAME                    STATUS   ROLES       AGE   VERSION    INTERNAL-IP   EXTERNAL-IP
   static-master-0         Ready    master      1h    v1.33.10   10.128.0.15   <none>
   yandex-worker-hybrid    Ready    yc-manual   5m    v1.33.10   10.128.0.17   <PUBLIC_IP>
   ```

8. If connection fails, check the NodeGroup status, events, and component logs:

```
d8 k get nodegroup yc-manual
d8 k describe nodegroup yc-manual
d8 k describe node yandex-worker-hybrid
d8 k -n d8-cloud-instance-manager logs deploy/machine-controller-manager --tail=200
```

Additional resources

- Global custom resources
- Module `cloud-provider-yandex` documentation
- Module `cni-cilium` parameters
- Module `node-manager` custom resources

Hybrid cluster with VCD

The following describes the process of adding nodes from VMware Cloud Director (VCD) to an existing static Deckhouse Kubernetes Platform (DKP) cluster.

Integration with VCD uses the `cloud-provider-vcd` module. It provides interaction between DKP and VMware Cloud Director, creation and deletion of virtual machines, retrieval of information about the VCD infrastructure, and integration with StorageClass and other provider capabilities.

This section describes two ways to add nodes:

- **Automatic node creation in VCD**. DKP creates virtual machines through the VCD API. VM parameters are defined by the VCDInstanceClass resource, and the required number of nodes is defined by the NodeGroup resource with the `CloudEphemeral` type.
- **Connecting manually created nodes through a bootstrap script**. A virtual machine is created by the user in advance and connected to the cluster using the DKP bootstrap script. This scenario uses the NodeGroup resource with the `CloudStatic` type.

Prerequisites for VCD

Before you begin, make sure that the following conditions are met:

- The cluster was created with the `clusterType: Static` parameter.
- Network connectivity is configured between the network of static nodes and the network of virtual machines in VCD. VCD nodes added to the cluster have access to the Kubernetes API, DNS, and the required addresses according to the Network interaction and Network policy configuration sections. When using Cilium with pod traffic tunneling, the `tunnelMode` mode is selected according to the network connectivity between sites.
- The requirements from the Connection and authorization in VMware vCloud Director section are met:
  - A tenant with allocated resources is configured in VCD.
  - A VCD account with a static password and administrator permissions is prepared.
  - A working network with an enabled DHCP server is configured in VCD.
  - The required VCD resources are prepared: VDC, vApp, templates, policies, and other parameters.

Adding automatically created nodes

1. Create a file with a ModuleConfig resource. For example, `cloud-provider-vcd-mc.yaml`:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: cloud-provider-vcd
   spec:
     version: 1
     enabled: true
     settings:
       mainNetwork: <NETWORK_NAME>
       organization: <ORGANIZATION>
       virtualDataCenter: <VDC_NAME>
       virtualApplicationName: <VAPP_NAME>
       sshPublicKey: <SSH_PUBLIC_KEY>
       provider:
         server: <API_URL>
         username: <USER_NAME>
         password: <PASSWORD>
         insecure: false
   ```

   Where:

   - `mainNetwork`: Name of the network where cloud nodes will be placed in VCD.
   - `organization`: Organization name in VCD.
   - `virtualDataCenter`: Virtual Data Center name in VCD.
   - `virtualApplicationName`: Name of the vApp where nodes will be created, for example `dkp-vcd-app`.
   - `sshPublicKey`: Public SSH key for accessing the nodes.
   - `provider.server`: VCD API URL.
   - `provider.username`: VCD username.
   - `provider.password`: VCD user password.
   - `provider.insecure`: Set to `true` if VCD uses a self-signed TLS certificate.

If a token is used for authentication, specify `apiToken` instead of `username` and `password`:

```
provider:
  server: <API_URL>
  apiToken: <API_TOKEN>
  insecure: false
```

2. Apply the ModuleConfig:

```
d8 k apply -f cloud-provider-vcd-mc.yaml
d8 k get mc cloud-provider-vcd
```

3. Make sure that all pods in the `d8-cloud-provider-vcd` namespace are in the `Running` state:

```
d8 k get pods -n d8-cloud-provider-vcd
```

4. Make sure that StorageClasses for VCD have been created in the cluster:

```
d8 k get sc
```

5. Create a file with the VCDInstanceClass and NodeGroup resources. For example, `vcd-instanceclass-nodegroup.yaml`:

```
---
apiVersion: deckhouse.io/v1
kind: VCDInstanceClass
metadata:
  name: worker
spec:
  rootDiskSizeGb: 50
  sizingPolicy: <SIZING_POLICY>
  storageProfile: <STORAGE_PROFILE>
  template: <VAPP_TEMPLATE>
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: VCDInstanceClass
      name: worker
    maxPerZone: 2
    minPerZone: 1
  nodeTemplate:
    labels:
      node-role/worker: ""
```

6. Apply the manifest:

```
d8 k apply -f vcd-instanceclass-nodegroup.yaml
```

After the manifest is applied, DKP will start creating virtual machines in VCD managed by the `node-manager` module.

7. Make sure that the required number of nodes has appeared in the cluster:

```
d8 k get nodes -o wide
```

8. If VM creation fails, check the Machine and MachineSet objects and the machine-controller-manager logs:

```
d8 k -n d8-cloud-instance-manager get machinesets,machines -o wide
d8 k -n d8-cloud-instance-manager logs deploy/machine-controller-manager --tail=200
```

Adding manually created nodes through a bootstrap script

Before you begin, make sure that the following conditions are met:

- The `cloud-provider-vcd` module is enabled:

```
d8 k get module cloud-provider-vcd -o wide
```

- The `cloud-provider-vcd` module components are in the `Running` state:

```
d8 k -n d8-cloud-provider-vcd get pods -o wide
```

- StorageClasses for VCD have been created in the cluster:

```
d8 k get sc
```

- A virtual machine that will be connected to the cluster has been created in VCD.
- The virtual machine name in VCD matches the hostname inside the operating system.
- The following value is set in the VM advanced parameters in VCD:

```
disk.EnableUUID = TRUE
```

- The virtual machine is connected to the VCD network used as the main network for the cluster's cloud nodes. Usually, this is the network specified in the `mainNetwork` parameter of the `cloud-provider-vcd` configuration or in the VCDInstanceClass being used.
- One of the package managers (`apt`/`apt-get`, `yum`, or `rpm`) for a supported OS is installed on the virtual machine.

1. Create a file with a NodeGroup resource and the CloudStatic node type. For example, `cloud-static-nodegroup.yaml`:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: cloud-static
spec:
  nodeType: CloudStatic
```

2. Make sure that the NodeGroup has been created and synchronized:

```
d8 k get nodegroup cloud-static
```

   Example expected output:

```
NAME           TYPE          READY   NODES   UPTODATE   INSTANCES   DESIRED   MIN   MAX
STANDBY   STATUS   AGE   SYNCED
cloud-static   CloudStatic   0       0       0
1m    True
```

3. Get the bootstrap script for the created NodeGroup:

```
NODE_GROUP=cloud-static

d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} \
  -o jsonpath='{.data.bootstrap\.sh}' > bootstrap.b64
```

4. Copy the bootstrap script to the virtual machine being connected:

```
scp bootstrap.b64 <USER>@<NODE_IP>:/tmp/bootstrap.b64
```

5. Connect to the virtual machine over SSH:

```
ssh <USER>@<NODE_IP>
```

6. On the virtual machine, set permissions and run the bootstrap script:

```
base64 -d /tmp/bootstrap.b64 > /tmp/bootstrap.sh
chmod +x /tmp/bootstrap.sh

sudo bash /tmp/bootstrap.sh
```

   After the bootstrap script is started, it will install the required components, configure the container runtime and kubelet, and connect the node to the cluster.

7. On the master node, check that the new node has appeared:

```
d8 k get nodes -o wide
```

   Example expected output:

```
NAME                      STATUS   ROLES          AGE   VERSION    INTERNAL-IP
static-master-0           Ready    master         1h    v1.33.10   192.168.240.138
cloud-static-worker-0     Ready    cloud-static   5m    v1.33.10   192.168.240.151
```

Additional resources

- Global custom resources
- Module `cloud-provider-vcd` documentation
- Module `cni-cilium` parameters
- Module `node-manager` custom resources

Hybrid cluster with vSphere

The following describes the process of adding nodes from vSphere to an existing static Deckhouse Kubernetes Platform (DKP) cluster.

Integration with vSphere uses the `cloud-provider-vsphere` module. It provides interaction between DKP and vCenter, retrieval of information about virtual machines, work with placement parameters, and integration with vSphere infrastructure capabilities.

This section describes two ways to add nodes:

- **Automatic node creation in vSphere**. DKP creates virtual machines through the vSphere API. VM parameters are defined by the VsphereInstanceClass resource, and the required number of nodes and placement zones are defined by the NodeGroup resource with the `CloudEphemeral` type.
- **Connecting manually created nodes through a bootstrap script**. A virtual machine is created by the user in advance and connected to the cluster using the DKP bootstrap script. This scenario uses the NodeGroup resource with the `CloudStatic` type.

Prerequisites for vSphere

Before you begin, make sure that the following conditions are met:

- The cluster was created with the `clusterType: Static` parameter.
- Network connectivity is configured between the network of static nodes and the network of virtual machines in vSphere.
- vSphere nodes added to the cluster have access to the Kubernetes API, DNS, and the required addresses according to the Network interaction and Network policy configuration sections.
- The requirements from the Connection and authorization in VMware vSphere section are met:
  - Access to vCenter is configured.
  - The vSphere account with the required privileges is prepared.
  - A virtual machine template is prepared.
  - Networks, Datastore, region tags, and zone tags are configured.
- When using Cilium with pod traffic tunneling, the `tunnelMode` mode is selected according to the network connectivity between sites.

Adding automatically created nodes

To connect an already running static cluster to vCenter, use the ModuleConfig resource of the `cloud-provider-vsphere` module.

In the `spec.settings` parameter, specify access parameters for vCenter, network settings, region and zone tags, and SSH keys that will be added to the created virtual machines.

An example configuration and description of the available parameters are provided in the module examples and in the section describing the module settings.

1. Create a file with ModuleConfig for the `cloud-provider-vsphere` module. For example, `vsphere-mc.yaml`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  version: 2
  enabled: true
  settings:
    host: "<VCENTER_FQDN>"
    username: "<USERNAME@DOMAIN.LOCAL>"
    password: "<PASSWORD>"
    insecure: true
    vmFolderPath: "<FOLDER_PATH_UNDER_DATACENTER>"
    regionTagCategory: "<TAG_CATEGORY_FOR_REGION>"
    zoneTagCategory: "<TAG_CATEGORY_FOR_ZONE>"
    region: "<REGION_TAG_NAME_ON_DATACENTER>"
    zones:
      - "<ZONE_TAG_NAME_ON_CLUSTER>"
    internalNetworkNames:
      - "<PORT_GROUP_NAME_FOR_INTERNAL_IP>"
    sshKeys:
      - "<SSH_PUBLIC_KEY_ONE_LINE>"
```

Parameter values:

- `host`: vCenter address.
- `username`, `password`: vSphere user credentials.
- `insecure`: Disables verification of the vCenter TLS certificate.
- `vmFolderPath`: Folder where virtual machines will be created.
- `regionTagCategory`, `zoneTagCategory`: Region and zone tag categories.
- `region`: Region tag.
- `zones`: List of zones where nodes can be created.
- `internalNetworkNames`: List of vSphere networks for connecting created nodes.
- `sshKeys`: Public SSH keys that will be added to the created virtual machines.

2. Apply the module configuration:

```
d8 k apply -f vsphere-mc.yaml
```

3. Wait for the `cloud-provider-vsphere` module to become ready:

```
d8 k get module cloud-provider-vsphere -o wide
```

4. Create a file with the VsphereInstanceClass and NodeGroup resources with the `nodeType: CloudEphemeral` value. For example, `vsphere-instance.yaml`:

```
apiVersion: deckhouse.io/v1
kind: VsphereInstanceClass
metadata:
  name: ephemeral
spec:
  numCPUs: 2
  memory: 4096
  rootDiskSize: 40
  template: "<PATH_TO_TEMPLATE_FROM_DATACENTER>"
  mainNetwork: "<PORT_GROUP_NAME>"
  datastore: "<DATASTORE_OR_FOLDER/DATASTORE>"
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ephemeral
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: VsphereInstanceClass
      name: ephemeral
    maxPerZone: 1
    minPerZone: 1
    zones:
      - "<ZONE_TAG_NAME_ON_CLUSTER>"
  disruptions:
    approvalMode: Automatic
```

Where:

- VsphereInstanceClass describes the parameters of the virtual machine that will be created in vSphere.
- NodeGroup describes the node group that DKP must maintain in the cluster.
- `nodeType: CloudEphemeral` means that nodes will be created automatically through the cloud provider.
- `cloudInstances.classReference` points to VsphereInstanceClass.
- `cloudInstances.zones` must contain zones from the `zones` list in ModuleConfig.

5. Apply the manifest:

   ```
   d8 k apply -f vsphere-instance.yaml
   ```

   After the manifest is applied, DKP will start creating a virtual machine in vSphere. After the VM boots, kubelet will connect to the Kubernetes API, and the new node will appear in the cluster.

6. Check the node status:

   ```
   d8 k get nodes -o wide
   ```

   Example expected output:

   ```
   NAME                              STATUS   ROLES                  AGE   VERSION
   static-master-0                   Ready    control-plane,master   1h    v1.33.10
   ephemeral-1ca02a5b-7588b-k89dc    Ready    ephemeral              10m   v1.33.10
   ```

7. If VM creation fails, check the Machine and MachineSet objects and the machine-controller-manager logs:

   ```
   d8 k -n d8-cloud-instance-manager get machinesets,machines -o wide
   d8 k -n d8-cloud-instance-manager logs deploy/machine-controller-manager --tail=200
   ```

   Also check the cluster events:

   ```
   d8 k get events -A --sort-by=.lastTimestamp | tail -n 100
   ```

Adding manually created nodes through a bootstrap script

Before you begin, make sure that the following conditions are met:

- The `cloud-provider-vsphere` module is enabled:

  ```
  d8 k get module cloud-provider-vsphere -o wide
  ```

- The `cloud-provider-vsphere` module components are in the `Running` state:

```
d8 k -n d8-cloud-provider-vsphere get pods -o wide
```

- StorageClasses for vSphere have been created in the cluster:

```
d8 k get sc
```

- A virtual machine that will be connected to the cluster has been created in vSphere.
- The virtual machine name in vSphere matches the hostname inside the operating system.
- The following parameters are set in the VM advanced parameters in vSphere:

```
disk.EnableUUID = TRUE
guestinfo.metadata = <BASE64_ENCODED_METADATA>
guestinfo.metadata.encoding = base64
```

The `guestinfo.metadata` parameter must contain the metadata configuration encoded in Base64. Example `metadata.json` file:

```
{
   "instance-id": "cloud-static-worker-0",
   "local-hostname": "cloud-static-worker-0",
   "public-keys-data": "<SSH_PUBLIC_KEY>",
   "network": {
     "version": 2,
     "ethernets": {
       "id0": {
         "match": {
           "driver": "vmxnet3"
         },
         "set-name": "ens192",
         "dhcp4": true
       }
     }
   }
 }
```

Where:

- `instance-id`: Virtual machine identifier.
- `local-hostname`: Node hostname inside the operating system.
- `public-keys-data`: Public SSH key for accessing the virtual machine.
- `network`: Network settings that will be applied inside the virtual machine.

To get the value for the `guestinfo.metadata` parameter, run:

```
METADATA_B64="$(base64 -w0 metadata.json)"
echo "$METADATA_B64"
```

- The virtual machine is connected to the network specified in the `internalNetworkNames` parameter of the `cloud-provider-vsphere` module configuration.
- One of the package managers (`apt`/`apt-get`, `yum`, or `rpm`) for a supported OS is installed on the virtual machine.

1. Create a file with a NodeGroup resource and the CloudStatic node type. For example, `cloud-static-nodegroup.yaml`:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: cloud-static
spec:
  nodeType: CloudStatic
```

2. Make sure that the NodeGroup has been created and synchronized:

```
d8 k get nodegroup cloud-static
```

Example expected output:

```
NAME           TYPE          READY   NODES   UPTODATE   INSTANCES   DESIRED   MIN   MAX
STANDBY   STATUS   AGE   SYNCED
cloud-static   CloudStatic   0       0       0
1m    True
```

3. Get the bootstrap script for the created NodeGroup:

```
NODE_GROUP=cloud-static

d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} \
  -o jsonpath='{.data.bootstrap\.sh}' > bootstrap.b64
```

4. Copy the bootstrap script to the virtual machine being connected:

```
scp bootstrap.b64 <USER>@<NODE_IP>:/tmp/bootstrap.b64
```

5. Connect to the virtual machine over SSH:

```
ssh <USER>@<NODE_IP>
```

6. On the virtual machine, set permissions and run the bootstrap script:

```
base64 -d /tmp/bootstrap.b64 > /tmp/bootstrap.sh
chmod +x /tmp/bootstrap.sh

sudo bash /tmp/bootstrap.sh
```

   After the bootstrap script is started, it will install the required components, configure the container runtime and kubelet, and connect the node to the cluster.

7. On the master node, check that the new node has appeared:

```
d8 k get nodes -o wide
```

   Example expected output:

```
NAME                      STATUS   ROLES          AGE   VERSION    INTERNAL-IP
static-master-0           Ready    master         1h    v1.33.10   192.168.240.135
cloud-static-worker-0     Ready    cloud-static   5m    v1.33.10   192.168.240.152
```

Additional resources

- Global custom resources
- Module `cloud-provider-vsphere` documentation
- Module `cni-cilium` parameters
- Module `node-manager` custom resources

Hybrid cluster with DVP

This section describes how to add nodes from Deckhouse Virtualization Platform (DVP) to an existing static DKP cluster.

Integration with DVP uses the `cloud-provider-dvp` module. It enables DKP to interact with the DVP cluster API, create virtual machines, connect the created VMs to an existing Kubernetes cluster, and manage the lifecycle of nodes through Cluster API mechanisms.

This section describes two ways to add nodes:

- **Automatically creating nodes in DVP**. DKP creates virtual machines through the DVP API. VM parameters are specified using the DVPInstanceClass resource, while the required number of nodes and placement zones are specified using the NodeGroup resource with the `CloudEphemeral` type.
- **Connecting manually created nodes using a bootstrap script**. A virtual machine is created in advance by the user and connected to the cluster using a DKP bootstrap script. This scenario uses a NodeGroup with the `CloudStatic` type.

Prerequisites

Before you begin, make sure that the following requirements are met:

- The cluster is created with the `clusterType: Static` parameter.
- Network connectivity is configured between the network of the static DKP cluster nodes and the network of virtual machines in DVP. For details, see Network requirements. Nodes created in DVP have access to the Kubernetes API of the target DKP cluster, DNS, and the required addresses according to the Network interaction and Configuring network policies sections. If Cilium is used with pod traffic tunneling, the `tunnelMode` mode is selected according to the network connectivity between sites. The Kubernetes API of the DVP cluster is accessible from the DKP cluster.
- The requirements from the Preparing the environment section are met:
  - a ServiceAccount has been created to access the DVP API;
  - a kubeconfig has been generated to connect to the DVP API;
  - a namespace has been prepared where virtual machines and disks will be created.
- A Linux OS image with `cloud-init` support is available in DVP, for example `ubuntu-24-04-lts`.
- A suitable VirtualMachineClass is available in DVP, for example `amd-epyc-gen-3`.
- A StorageClass for root disks of virtual machines is available in DVP, for example `replicated`.
- If a virtual machine template is used, make sure that it contains only one disk.

⚠ DVPInstanceClass parameters use resources from the DVP cluster: VirtualMachineClass, ClusterVirtualImage, VirtualImage, VirtualDisk, and StorageClass from DVP, not from the target DKP cluster.

⚠ Make sure there are no StorageClass resources in the static DKP cluster whose names match the names of StorageClass resources in the DVP cluster.

When the `cloud-provider-dvp` module is enabled, the corresponding StorageClass resources are automatically synchronized from DVP to the DKP cluster. If a StorageClass with the same name already exists in the static cluster, a resource conflict may occur, causing the module installation or upgrade to fail.

Adding automatically created nodes

1. On the administrator's machine where access to the DVP cluster is configured, prepare a kubeconfig for the `cloud-provider-dvp` module to access the DVP API.

   Follow the steps in the "Preparing the environment" section and encode the generated kubeconfig in Base64:

   ```
   export DVP_PROVIDER_KUBECONFIG="./kubeconfig"
   export DVP_KUBECONFIG_B64="$(base64 -w0 ${DVP_PROVIDER_KUBECONFIG})"
   ```

2. Specify the DVP namespace where virtual machines and disks will be created:

   ```
   export DVP_NAMESPACE="<DVP_NAMESPACE>"
   ```

3. Specify the DVP zone where nodes will be created.

   Currently, zoning in DVP is under development, so use the `default` value for the `zones` parameters in ModuleConfig and NodeGroup:

   ```
   export DVP_ZONE="default"
   ```

   If necessary, you can check the topology labels of nodes in the DVP cluster:

   ```
   d8 k get nodes -L topology.kubernetes.io/region,topology.kubernetes.io/zone
   ```

   ⚠ The zone value in ModuleConfig and NodeGroup must match. Currently, only the `default` value is available in DVP.

4. Create a file with the `cloud-provider-dvp` module configuration. For example, `cloud-provider-dvp-mc.yaml`:

   ```
   cat > cloud-provider-dvp-mc.yaml <<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: cloud-provider-dvp
   spec:
     enabled: true
     version: 1
     settings:
       provider:
         kubeconfigDataBase64: ${DVP_KUBECONFIG_B64}
         namespace: ${DVP_NAMESPACE}
       zones:
         - ${DVP_ZONE}
   EOF
   ```

   The manifest automatically uses the values of the environment variables set in the previous steps: `DVP_KUBECONFIG_B64`, `DVP_NAMESPACE`, and `DVP_ZONE`.

5. Apply ModuleConfig:

   ```
   d8 k apply -f cloud-provider-dvp-mc.yaml
   ```

6. Wait until the `cloud-provider-dvp` module is enabled:

   ```
   d8 k get module cloud-provider-dvp -o wide
   ```

   The module must switch to the `Ready` state, and the pods in the `d8-cloud-provider-dvp` namespace must be in the `Running` state.

7. Make sure that the `node-manager` module is in the `Ready` state:

   ```
   d8 k get module node-manager -o wide
   ```

   If the module is in the `Error` state, check that available DVP zones are specified in ModuleConfig and NodeGroup.

8. Make sure that the DVPInstanceClass resource is available in the cluster:

   ```
   d8 k get crd dvpinstanceclasses.deckhouse.io
   ```

9. On the administrator's machine where access to the DVP cluster is configured, check the available virtual machine classes, images, and StorageClasses:

   ```
   d8 k --kubeconfig ${DVP_PROVIDER_KUBECONFIG} get virtualmachineclasses
   d8 k --kubeconfig ${DVP_PROVIDER_KUBECONFIG} get clustervirtualimages
   d8 k --kubeconfig ${DVP_PROVIDER_KUBECONFIG} get storageclasses
   ```

   Use the obtained values when creating DVPInstanceClass.

10. Create a file with DVPInstanceClass and NodeGroup resources. For example, `dvp-instanceclass-nodegroup.yaml`:

    ```
    apiVersion: deckhouse.io/v1alpha1
    kind: DVPInstanceClass
    metadata:
      name: dvp-worker
    spec:
      virtualMachine:
        cpu:
          cores: 3
          coreFraction: 20%
        memory:
          size: 6Gi
        virtualMachineClassName: <VIRTUAL_MACHINE_CLASS_NAME>
        bootloader: EFI
      rootDisk:
        size: 15Gi
        storageClass: <STORAGE_CLASS_NAME>
        image:
          kind: ClusterVirtualImage
          name: <CLUSTER_VIRTUAL_IMAGE_NAME>
    ---
    apiVersion: deckhouse.io/v1
    kind: NodeGroup
    metadata:
      name: dvp-worker
    spec:
      nodeType: CloudEphemeral
      cloudInstances:
        classReference:
          kind: DVPInstanceClass
          name: dvp-worker
        minPerZone: 1
        maxPerZone: 1
        zones:
          - default
    ```

    Where:

    - `virtualMachineClassName` — name of the VirtualMachineClass in DVP, for example `amd-epyc-gen-3`;
    - `rootDisk.storageClass` — name of the StorageClass in DVP, for example `replicated`;
    - `rootDisk.image.kind` — image source type. For a cluster image, use `ClusterVirtualImage`;
    - `rootDisk.image.name` — name of the OS image in DVP, for example `ubuntu-24-04-lts`;
    - `cloudInstances.zones` — DVP zone where the node will be created. The value must match the `zones` value in ModuleConfig.

11. Apply the manifest:

    ```
    d8 k apply -f dvp-instanceclass-nodegroup.yaml
    ```

    After the manifest is applied, DKP will start creating a virtual machine in DVP and connect it to the cluster as a node.

12. Check the NodeGroup status:

    ```
    d8 k get nodegroup dvp-worker -o wide
    d8 k describe nodegroup dvp-worker
    ```

13. Check that a new node appears in the DKP cluster:

    ```
    d8 k get nodes -o wide
    ```

    Example expected output:

    ```
    NAME                              STATUS   ROLES                  AGE   VERSION    INTERNAL-IP
    dvp-hybrid-master-0               Ready    control-plane,master   1h    v1.33.10   10.12.0.69
    dvp-worker-c75a75c1-twqp4-bjpvl   Ready    dvp-worker             10m   v1.33.10   10.12.3.15
    ```

Adding manually created nodes using a bootstrap script

Before you begin, make sure that the following requirements are met:

- The `cloud-provider-dvp` module is enabled:

  ```
  d8 k get module cloud-provider-dvp -o wide
  ```

- The `cloud-provider-dvp` module components are in the `Running` state:

  ```
  d8 k -n d8-cloud-provider-dvp get pods -o wide
  ```

- A virtual machine that will be connected to the cluster has been created in DVP.
- The virtual machine is connected to the DVP network used for hybrid integration with the cluster.
- The virtual machine IP address belongs to the range specified in `internalNetworkCIDRs`.
- The virtual machine name in DVP matches the hostname inside the operating system.
- SSH access is available on the virtual machine for copying and running the bootstrap script.
- The SSH user can run commands using `sudo` without entering a password.
- One of the package managers (`apt`/`apt-get`, `yum`, or `rpm`) for a supported OS is installed on the virtual machine.

1. Create a NodeGroup with the `CloudStatic` node type. In this example and the following steps, the `cloud-static` name is used:

   ```
   d8 k apply -f - <<EOF
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: cloud-static
   spec:
     nodeType: CloudStatic
   EOF
   ```

2. Make sure that NodeGroup has been created and synchronized:

   ```
   d8 k get nodegroup cloud-static
   ```

   Example expected output:

   ```
   NAME           TYPE          READY   NODES   UPTODATE   INSTANCES   DESIRED   MIN   MAX
   STANDBY   STATUS   AGE   SYNCED
   cloud-static   CloudStatic   0       0       0
   1m    True
   ```

3. Get the bootstrap script for the created NodeGroup:

   ```
   NODE_GROUP=cloud-static

   d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} \
     -o jsonpath='{.data.bootstrap\.sh}' > ${NODE_GROUP}-bootstrap.b64
   ```

4. On the master node, verify that the file contains valid Base64 data of the bootstrap script:

   ```
   base64 -d ${NODE_GROUP}-bootstrap.b64 > /dev/null
   ```

   Check the beginning of the decoded content:

   ```
   base64 -d ${NODE_GROUP}-bootstrap.b64 | head -n 5
   ```

   The decoded content must start with a bash script:

   ```
   #!/bin/bash
   ...
   ```

5. Copy the bootstrap script to the virtual machine being connected:

   ```
   scp ${NODE_GROUP}-bootstrap.b64 <USER>@<NODE_IP>:/tmp/bootstrap.b64
   ```

6. Connect to the virtual machine over SSH:

   ```
   ssh <USER>@<NODE_IP>
   ```

7. On the virtual machine, decode the bootstrap script, set permissions, and run it:

   ```
   base64 -d /tmp/bootstrap.b64 > /tmp/bootstrap.sh
   chmod +x /tmp/bootstrap.sh

   sudo bash /tmp/bootstrap.sh
   ```

   After the bootstrap script starts, it will install the required components, configure the container runtime and kubelet, and connect the node to the cluster.

8. On the master node, check that a new node appears:

   ```
   d8 k get nodes -o wide
   ```

   Example expected output:

   ```
   NAME                  STATUS   ROLES                  AGE   VERSION    INTERNAL-IP
   dvp-hybrid-master-0   Ready    control-plane,master   1h    v1.33.12   10.12.0.69
   cloud-static-worker-0 Ready    cloud-static           5m    v1.33.12   10.12.3.88
   ```

9. If connection fails, check the NodeGroup status, events, and bootstrap logs on the virtual machine being connected:

   ```
   d8 k get nodegroup cloud-static
   d8 k describe nodegroup cloud-static
   d8 k get events -A --sort-by=.lastTimestamp | tail -n 100
   ```

   On the virtual machine being connected:

   ```
   sudo tail -n 120 /var/log/d8/bashible/bootstrap.log
   ```

   If the logs contain the `Failed to discover node ip that matches internalNetworkCIDRs` error, check that the virtual machine IP address belongs to `internalNetworkCIDRs`.

Additional resources

- Global custom resources
- Module `cloud-provider-dvp` documentation
- Module `cni-cilium` parameters
- Module `node-manager` custom resources
- Module `virtualization` documentation

High reliability and availability

The reliability and resilience of a Kubernetes cluster are key characteristics that define the stability of the infrastructure. Deckhouse Kubernetes Platform (DKP) ensures high availability (HA) and fault tolerance through built-in mechanisms and modules.

When HA mode is enabled, critical cluster components are launched with the necessary redundancy to guarantee their continuous operation. Even if a single instance fails, services can continue functioning without downtime. For more information on enabling HA mode, refer to Managing HA mode.

If the cluster has more than one master node, HA mode is automatically enabled, both during the initial deployment and when additional nodes are added later. Recommended roles and number of nodes can be found in Recommendations for configuring cluster nodes and preventing overload.

DKP provides chaos engineering tools to test cluster resilience. These tools let you deliberately or randomly disrupt individual components and observe the infrastructure’s response. For information on configuring these tools, refer to Chaos engineering.

You can further increase cluster fault tolerance by enabling inter-cluster communication based on the Service Mesh mode of the `istio` module. In this mode, federation is configured between multiple clusters. In case of failures in one cluster, the load is automatically redistributed to others. For configuration details, refer to Federation.

Additional resources

- Module `istio` documentation

Recommendations for configuring cluster nodes and preventing overload

Proper node configuration increases cluster fault tolerance and simplifies maintenance. Below are the recommended roles and number of nodes for various components.

Master nodes

To ensure cluster fault tolerance, **always** use at least three master nodes. This number ensures uninterrupted cluster operation and lets you update master nodes safely. More than three nodes are unnecessary, and two nodes are insufficient to maintain alignment (quorum) in case of issues in one of the nodes.

Using only one master node means its failure will bring down the entire cluster since it manages key components that keep the cluster running.

Frontend nodes

Frontend nodes balance incoming traffic. Ingress controllers run on them. Use more than one frontend node. These nodes must be able to handle traffic in case at least one frontend node fails.

For example, if the cluster has two frontend nodes, each must be capable of handling the full load if the other fails. If there are three, each node must be able to handle at least a 150% load increase.

Monitoring nodes

Monitoring nodes run Grafana, Prometheus, and other monitoring components. In heavily loaded clusters with multiple alerts and large metric volumes, it’s recommended that you allocate dedicated nodes for monitoring. Otherwise, monitoring components will run on system nodes.

When dedicating nodes for monitoring, it’s important they use fast disks (at least 400 IOPS).

System nodes

System nodes are used to run Deckhouse modules. Allocate two system nodes. This ensures Deckhouse modules will run on them without interfering with user applications in the cluster.

Preventing Kubernetes API overload (FlowSchema)

By default, a DKP cluster includes a component that implements FlowSchema and PriorityLevelConfiguration to prevent Kubernetes API overload.

`FlowSchema` assigns `PriorityLevel` for `list` requests from all service accounts in Deckhouse namespaces (with the label `heritage: deckhouse`) to the following apiGroups:

- `v1` (Pod, Secret, ConfigMap, Node, etc.): Useful for clusters with numerous basic resources (for example, Secrets or Pods).
- `apps/v1` (DaemonSet, Deployment, StatefulSet, ReplicaSet, etc.): Useful when numerous applications are deployed in the cluster (for example, Deployments).
- `deckhouse.io` (Deckhouse custom resources): Useful when there are numerous Deckhouse custom resources in the cluster.
- `cilium.io` (Cilium custom resources): Useful for clusters with numerous Cilium policies.

All API requests matching a `FlowSchema` are placed into a single queue with a shared priority. This limits the load on the Kubernetes API and prevents it from being overloaded by mass `list` requests.

This component does not have settings, but the following commands are available:

- Check the state of priority levels:

  ```
  d8 k get --raw /debug/api_priority_and_fairness/dump_priority_levels
  ```

- Check the state of priority level queues:

  ```
  d8 k get --raw /debug/api_priority_and_fairness/dump_queues
  ```

The component also provides the following metrics to Grafana:

- `apiserver flowcontrol rejected requests total`: Total number of rejected requests.
- `apiserver flowcontrol dispatched requests total`: Total number of processed requests.
- `apiserver flowcontrol current inqueue requests`: Number of requests in queues.
- `apiserver_flowcontrol_current_executing_requests`: Number of requests being executed.

Delaying node reboot or shutdown while critical pods are running

DKP supports a mechanism for delaying the shutdown or reboot of a node if it is running stateful applications or virtual machines (VMs). This is implemented by the `d8-shutdown-inhibitor` service of the `node-manager` module.

The delay is triggered if pods with the label `pod.deckhouse.io/inhibit-node-shutdown` are running on the node. DKP delays the shutdown or restart of the node so that the user can migrate critical pods or terminate them in a controlled manner. This is useful, for example, for stateful workloads and other cases where data loss must be avoided. For information about the mechanism’s principles and features, refer to the `node-manager` module documentation.

Enabling a node shutdown delay

⚠ To decide whether to block a node shutdown, DKP additionally queries the NodeGroup. If the current node belongs to the `master` group and it is the only master node in the cluster, the shutdown block will not be applied to it.

To enable the mechanism that delays a node's restart or shutdown, add the label `pod.deckhouse.io/inhibit-node-shutdown` to the Pod (for a Deployment, specify the label in the pod template).

Example of a Pod manifest with the `pod.deckhouse.io/inhibit-node-shutdown` label:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-stateful-app
  labels:
    app: my-stateful-app
    pod.deckhouse.io/inhibit-node-shutdown: "true"
spec:
  containers:
    - name: app
      image: my-registry/my-app:1.0.0
```

⚠ The block only applies to pods labeled with `pod.deckhouse.io/inhibit-node-shutdown` that are in the `Running` phase on the node. Kubelet shuts down the pods without this label according to the standard graceful shutdown process.

Searching pods that prevent shutdown and performing the node troubleshooting

To search the pods that are blocking a node shutdown, use the following command:

```
d8 k get po -A -l pod.deckhouse.io/inhibit-node-shutdown -o wide
```

To view the `GracefulShutdownPostpone` condition on a node, use the following command:

```
d8 k get node <NODE_NAME> -o jsonpath='{range .status.conditions[?(@.type=="GracefulShutdownPostpone")]}
{.type}{"\t"}{.status}{"\t"}{.reason}{"\t"}{.message}{"\n"}{end}'
```

Possible values of the `reason` field:

- `WaitingForShutdownSignal`: The shutdown blocking mechanism is running on the node, but the blocking has not yet begun.
- `PodsWithLabelAreRunningOnNode`: The shutdown has been blocked because there are pods with the label `pod.deckhouse.io/inhibit-node-shutdown` still running on the node.
- `NoRunningPodsWithLabel`: The blocking has been released and the shutdown can continue.

Additional resources

- Module `node-manager` documentation

Managing HA mode

ℹ Note that if the cluster has **more than one master node**, HA mode is **enabled automatically**. This applies both when deploying a cluster with multiple master nodes from the start and when increasing the number of master nodes from one to three.

Enabling HA mode globally

You can enable HA mode globally for DKP in one of the following ways.

Using ModuleConfig/global custom resource

1. Set the `settings.highAvailability` parameter to `true` in `ModuleConfig/global`:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: global
   spec:
     version: 2
     settings:
       highAvailability: true
   ```

2. To ensure HA mode is enabled, you can, for example, check the number of `deckhouse` Pods in the `d8-system` namespace. To do that, run the following command:

   ```
   d8 k -n d8-system get po | grep deckhouse
   ```

   The number of `deckhouse` Pods in the output must be more than one:

   ```
   deckhouse-57695f4d68-8rk6l                        2/2     Running   0             3m49s
   deckhouse-5764gfud68-76dsb                        2/2     Running   0             3m49s
   deckhouse-fgrhy4536s-fhu6s                        2/2     Running   0             3m49s
   ```

Using Deckhouse web UI

If the `console` module is enabled in the cluster, open the Deckhouse web UI, navigate to **Deckhouse** — **Global settings** — **Global module settings**, and switch the **HA mode** toggle to **Yes**.

Configuring HA mode with two master nodes and an arbiter node

Deckhouse Kubernetes Platform allows you to configure HA mode with two master nodes and an arbiter node. This approach allows you to meet HA requirements in conditions of limited resources.

Only etcd is placed on the arbiter node, without the other control plane components. This node is used to ensure the etcd quorum.

Requirements for the arbiter node:

- At least 2 CPU cores
- At least 4 GB of RAM
- At least 8 GB of disk space for etcd

The network latency requirements for the arbiter node are similar to those for the master nodes.

Configuring in a cloud cluster

The example below applies to a cloud cluster with three master nodes. To configure HA mode with two master nodes and an arbiter node in a cloud cluster, you need to remove one master node from the cluster and add one arbiter node.

To do this, follow these steps:

⚠ The following steps must be performed starting from the first master node (`master-0`) in the cluster. This is because the cluster scales in order — for example, it is not possible to remove `master-0` and `master-1` while leaving `master-2`.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. We strongly recommend creating a backup of the module's data before making any changes.

1. Create a backup of etcd and the `/etc/kubernetes` directory.
2. Copy the resulting archive outside the cluster (for example, to a local machine).
3. Ensure there are no alerts in the cluster that may interfere with the master node update process.
4. Make sure the DKP queue is empty:

   ```
   d8 system queue list
   ```

5. On the **local machine**, run the DKP installer container for the corresponding edition and version (change the container registry address if needed):

   ```
   DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
   DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]' )
   docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
     registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
   ```

6. **In the installer container**, run the following command:

   ```
   dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
     --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
   ```

   Change the cloud provider settings:

   - Set `masterNodeGroup.replicas` to `2`.
   - Create a NodeGroup for the arbiter node. The arbiter node **must have** the label `node.deckhouse.io/etcd-arbiter: ""` and a taint that prevents user workloads from being placed on it. Example of a NodeGroup description for the arbiter node:

```
nodeGroups:
  - name: arbiter
    replicas: 1
    nodeTemplate:
      labels:
        node.deckhouse.io/etcd-arbiter: ""
      taints:
        - key: node.deckhouse.io/etcd-arbiter
          effect: NoSchedule
    zones:
      - europe-west3-b
    instanceClass:
      machineType: n1-standard-4
  # ... the rest of the manifest
```

- Save your changes.

> **i** For **Yandex Cloud**, if external IPs are used for master nodes, the number of items in the `masterNodeGroup.instanceClass.externalIPAddresses` array must match the number of master nodes. Even when using `Auto` (automatic public IP allocation), the number of entries must still match.
>
> For example, for a single master node (`masterNodeGroup.replicas: 1`) and automatic IP assignment, the `masterNodeGroup.instanceClass.externalIPAddresses` section would look as follows:
>
> ```
> externalIPAddresses:
> - "Auto"
> ```

7. **In the installer container**, run the following command to trigger the scaling operation:

   ```
   dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host
   <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST>
   ```

   > **i** **Important**. For **OpenStack** and **VK Cloud (OpenStack)**, after confirming the node deletion, it is extremely important to check the disk deletion `<prefix>kubernetes-data-N` in OpenStack itself.
   >
   > For example, when deleting the `cloud-demo-master-2` node in the OpenStack web interface or in the OpenStack CLI, it is necessary to check the absence of the `cloud-demo-kubernetes-data-2` disk.
   >
   > If the `kubernetes-data` disk remains, there may be problems with etcd operation as the number of master nodes increases.

8. Check the Deckhouse queue with the following command and make sure there are no errors:

   ```
   d8 system queue list
   ```

Configuring in a static cluster

To configure HA mode with two master nodes and an arbiter node in a static cluster, follow these steps:

1. Create a NodeGroup for the arbiter node. The arbiter node **must have** the label `node.deckhouse.io/etcd-arbiter: “”` and a taint that prevents user workloads from being placed on it. Example of a NodeGroup description for the arbiter node:

   ```
   apiVersion: deckhouse.io/v1
     kind: NodeGroup
     metadata:
       name: arbiter
     spec:
       nodeType: Static
       nodeTemplate:
         labels:
           node.deckhouse.io/etcd-arbiter: ""
         taints:
           - key: node.deckhouse.io/etcd-arbiter
             effect: NoSchedule
     # ... the rest of the manifest
   ```

2. Add a node to the cluster that will be used as an arbiter node in a way that is convenient for you.
3. Ensure that the added arbiter node is in the list of etcd cluster members.
4. Remove one master node from the cluster.

Enabling HA mode for individual components

Some DKP modules may have their own HA mode settings. To enable HA mode in a specific module, set the `settings.highAvailability` parameter in its configuration. The HA mode operation in individual modules is independent of the global HA mode.

List of modules supporting individual HA mode:

- `deckhouse`
- `openvpn`
- `istio`
- `dashboard`
- `multitenancy-manager`
- `user-authn`
- `ingress-nginx`
- `prometheus-monitoring`
- `monitoring-kubernetes`
- `snapshot-controller`

To enable HA mode manually for a specific module, add the `settings.highAvailability` parameter to its configuration:

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

To ensure HA mode is enabled, check the number of Pods for the target module. For example, to verify the mode operation for the `deckhouse` module, check the number of corresponding Pods in the `d8-system` namespace by running the following command:

```
d8 k -n d8-system get po | grep deckhouse
```

The number of `deckhouse` Pods in the output must be more than one:

```
deckhouse-57695f4d68-8rk6l                        2/2     Running   0             3m49s
deckhouse-5764gfud68-76dsb                        2/2     Running   0             3m49s
deckhouse-fgrhy4536s-fhu6s                        2/2     Running   0             3m49s
```

Additional resources

- Global parameters
- Module `console` documentation
- Module `control-plane-manager` documentation
- Module `deckhouse` documentation
- Module `openvpn` documentation
- Module `stronghold` documentation

Chaos engineering

> ⚠ Chaos engineering mode can only be enabled for node groups with `nodeType: CloudEphemeral`.

Enable chaos engineering mode for a node group in one of the following ways:

1. Define the `spec.chaos` parameter in the configuration of the target node group with two nested parameters:

   ```
   chaos:
     mode: DrainAndDelete
     period: 24h
   ```

   Where:

   - `mode`: Operation mode. Two options are available:
     - `DrainAndDelete`: When triggered, drains the node and deletes it.
     - `Disabled`: Skips this specific node group.
   - `period`: Chaos Monkey trigger interval. Defined as a string with hours and minutes: `30m`, `1h`, `2h30m`, `24h`.

   Example configuration for a node group:

```
# NodeGroup for cloud nodes in AWS.
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: test
spec:
  nodeType: CloudEphemeral
  chaos:
    mode: DrainAndDelete
    period: 24h
```

2. If the `console` module is enabled in the cluster, open the Deckhouse web UI, go to the settings of the desired node group under **Nodes** — **Node Groups**, and enable Chaos Monkey in the **Chaos monkey settings** section by specifying the time intervals in the corresponding fields.

Additional resources

- Module `console` documentation
- Module `node-manager` custom resources

Platform scaling and node management

Deckhouse Kubernetes Platform (DKP) provides built-in mechanisms for comprehensive management of cluster architecture — both at the control plane level and at the node level.

Management capabilities include:

- Control plane management — automation of configuration and updates for Kubernetes API, etcd, and other system components; issuance and renewal of certificates; scaling of the control plane and transitioning between single-master and multi-master modes; high availability and recovery.
- Node management — creation and scaling of node groups using NodeGroup objects; automatic deployment and updating of nodes; support for various types of infrastructure (cloud and bare-metal); management of static and cloud nodes; use of additional tools for fine-tuning.

These capabilities allow the creation of reliable, scalable, and self-healing clusters, adapting them to any requirements for performance, high availability, and infrastructure constraints.

The following sections provide detailed descriptions of features, configuration examples, and best practices for effective control plane and node management in Deckhouse Kubernetes Platform.

General management and configuration of the control plane

Main features

Deckhouse Kubernetes Platform (DKP) manages control plane components using the `control-plane-manager` module, which runs on all master nodes (nodes with the label `node-role.kubernetes.io/control-plane: ""`).

The control plane management functionality includes:

- Managing certificates required for the control plane to function, including their renewal and issuance when the configuration changes. Secure configuration is maintained automatically, with the ability to quickly add additional SANs for secure access to the Kubernetes API.
- Component configuration. DKP generates all necessary configurations and manifests (kube-apiserver, etcd, etc.), reducing the risk of human error.
- Upgrade/downgrade of components. DKP supports consistent version upgrades or downgrades of the control plane, helping maintain version uniformity across the cluster.
- Managing the etcd cluster configuration and its members. DKP scales master nodes and performs migrations between single-master and multi-master modes.
- Configuring kubeconfig. DKP generates an up-to-date configuration file (with `cluster-admin` permissions), creates, renews, and updates the kubeconfig for control-plane components and the admin kubeconfig (`admin.conf`), and creates a symlink for the `root` user (`/root/.kube/config` -> `admin.conf`). When the `user-authz` module is enabled, the symlink can be disabled using the `rootKubeconfigSymlink` parameter in the `control-plane-manager` module (for more details, see the FAQ of the `control-plane-manager` module). DKP also tightens access permissions for the `admin.conf` and `super-admin.conf` files to enhance security.

> **i** Some parameters affecting control plane behavior are taken from the ClusterConfiguration resource.

Enabling, disabling, and configuring the module

Enabling / disabling

You can enable or disable the `control-plane-manager` module in the following ways:

1. Create (or modify) the ModuleConfig/control-plane-manager resource by setting `spec.enabled` to `true` or `false`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  enabled: true
```

2. Using the command:

   ```
   d8 system module enable control-plane-manager
   ```

   or to disable:

   ```
   d8 system module disable control-plane-manager
   ```

3. Via the Deckhouse web interface:
   - Go to the “Deckhouse → Modules” section.
   - Find the `control-plane-manager` module and click on it.
   - Toggle the “Module enabled” switch.

Configuration

To configure the module, use the ModuleConfig/control-plane-manager resource and specify the required parameters in `spec.settings`.

Example with the schema version, enabled module, and a few settings:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 1
  enabled: true
  settings:
    apiserver:
      bindToWildcard: true
      certSANs:
      - bakery.infra
      - devs.infra
      loadBalancer: {}
```

Checking DKP status and queues

How to verify that `control-plane-manager` is running correctly and is not in a pending state, and how to check active DKP tasks (queues):

1. Make sure the module is enabled:

   ```
   d8 k get modules control-plane-manager
   ```

2. Check the status of `control-plane-manager` pods (they run in the `kube-system` namespace and have the label `app=d8-control-plane-manager`):

   ```
   d8 k -n kube-system get pods -l app=d8-control-plane-manager -o wide
   ```

   Ensure that all pods are in the `Running` or `Completed` state.

3. Verify that master nodes are in the `Ready` state:

   ```
   d8 k get nodes -l node-role.kubernetes.io/control-plane
   ```

   To view detailed information:

   ```
   d8 k describe node <node-name>
   ```

4. Get the list of queues and active tasks:

   ```
   d8 system queue list
   ```

   Example output:

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

⚠ Before performing heavy operations (e.g., transitioning from single-master to multi-master or upgrading the Kubernetes version), it is recommended to wait until all tasks in the queues are completed.

Certificate management

In DKP, the `control-plane-manager` module is responsible for issuing and renewing all SSL certificates for control plane components. It manages:

1. **Server certificates** for kube-apiserver and etcd, stored in the `d8-pki` secret (namespace: `kube-system`):
   - Kubernetes root CA (`ca.crt`, `ca.key`).
   - etcd root CA (`etcd/ca.crt`, `etcd/ca.key`).
   - RSA certificate and key for signing Service Accounts (`sa.pub`, `sa.key`).
   - Root CA for extension API servers (`front-proxy-ca.key`, `front-proxy-ca.crt`).
2. **Client certificates** required for mutual communication between control plane components (e.g., `apiserver.crt`, `apiserver-etcd-client.crt`, etc.). These files are stored only on the nodes. If any changes occur (e.g., new SANs are added), certificates are automatically reissued, and the kubeconfig is synchronized.

PKI management

DKP also manages the Public Key Infrastructure (PKI) used for encryption and authentication throughout the Kubernetes cluster:

- PKI for control plane components (kube-apiserver, kube-controller-manager, kube-scheduler, etc.).
- PKI for the etcd cluster (etcd certificates and inter-node communication).

DKP assumes control of this PKI after the initial cluster installation and once its pods are running. As a result, all key issuance, renewal, and rotation operations (both for control plane and etcd) are performed automatically and centrally, without requiring manual intervention.

Additional SANs and auto-update

Deckhouse simplifies the addition of new Subject Alternative Names (SANs) for the Kubernetes API endpoint: you only need to specify them in the configuration. After any SAN change, the module automatically regenerates the certificates and updates the kubeconfig.

To add additional SANs (extra DNS names or IP addresses) for the Kubernetes API, specify the new SANs in the `spec.settings.apiserver.certSANs` field of your ModuleConfig/control-plane-manager resource.

DKP will automatically generate new certificates and update all required configuration files (including kubeconfig).

Kubelet certificate rotation

In Deckhouse Kubernetes Platform, kubelet certificate rotation is automatic. The `--tls-cert-file` and `--tls-private-key-file` parameters for kubelet are not set directly. Instead, a dynamic TLS certificate mechanism is used: kubelet applies the client certificate located at `/var/lib/kubelet/pki/kubelet-client-current.pem`, which it uses to request a new client or server certificate (file `/var/lib/kubelet/pki/kubelet-server-current.pem`) from kube-apiserver. Also, the CIS benchmark `AVD-KCV-0088` and `AVD-KCV-0089` checks, which track whether the `--tls-cert-file` and `--tls-private-key-file` arguments were passed to kubelet, are disabled in the `operator-trivy` module.

Features of kubelet certificate rotation in Deckhouse Kubernetes Platform:

- By default, kubelet generates its own keys in the `/var/lib/kubelet/pki/` directory and, if necessary, independently requests certificate renewal from kube-apiserver.
- lifetime of certificates is 1 year (8760 hours). When there are between 5 and 10% of the time remaining before expiration (the exact value is randomly selected from this range), kubelet automatically initiates a request for a new certificate. For more details, see the official documentation Kubernetes. If necessary, lifetime of certificates can be changed using `--cluster-signing-duration` parameter in `/etc/kubernetes/manifests/kube-controller-manager.yaml` manifest. However, in order for kubelet to obtain and install a new certificate before the current one expires, it is recommended to set the validity period of certificates to at least 1 hour.
- If the kubelet fails to renew the certificate before it expires, it will lose the ability to make requests to the kube-apiserver and, accordingly, renew certificates. As a result, the node will be marked as `NotReady` and automatically recreated.

Manual renewal of control plane certificates

If the master nodes were offline for an extended period (e.g., the servers were shut down), some control plane certificates may become outdated. In such cases, automatic renewal will not occur upon reboot — manual intervention is required.

To manually renew the control plane certificates, use the `kubeadm` utility on each master node:

1. Locate the `kubeadm` binary on the master node and create a symbolic link:

   ```
   ln -s  $(find /var/lib/containerd  -name kubeadm -type f -executable -print -quit) /usr/bin/kubeadm
   ```

2. Execute the following command:

```
kubeadm certs renew all
```

This command will regenerate the necessary certificates (for kube-apiserver, kube-controller-manager, kube-scheduler, etcd, and others).

Speeding up pod restarts after losing connection to a node

By default, if a node does not report its status within 40 seconds, it is marked as `Unreachable`. Then, after another 5 minutes, the pods on that node begin to be restarted on other nodes. As a result, the total application downtime can reach approximately 6 minutes.

In specific cases where an application cannot run in multiple instances, there is a way to reduce the downtime period:

1. Reduce the time before a node is marked as `Unreachable` by configuring the `nodeMonitorGracePeriodSeconds` parameter.
2. Set a shorter timeout for evicting pods from the unreachable node using the `failedNodePodEvictionTimeoutSeconds` parameter.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 1
  settings:
    nodeMonitorGracePeriodSeconds: 10
    failedNodePodEvictionTimeoutSeconds: 50
```

⚠ The shorter the timeouts, the more frequently system components check node status and plan pod rescheduling. This increases the load on the control plane, so choose values that match your requirements for high availability and performance.

Forcibly disabling IPv6 on cluster nodes

Internal communication between Deckhouse cluster components is performed via IPv4 protocol. However, at the operating system level of the cluster nodes, IPv6 is usually active by default. This leads to automatic assignment of IPv6 addresses to all network interfaces, including Pod interfaces. This results in unwanted network traffic - for example, redundant DNS queries like `AAAAA` - which can affect performance and make debugging network communications more difficult.

To correctly disable IPv6 at the node level in a Deckhouse-managed cluster, it is sufficient to set the necessary parameters via the NodeGroupConfiguration resource:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: disable-ipv6.sh
spec:
  nodeGroups:
  - '*'
  bundles:
  - '*'
  weight: 50
  content: |
    GRUB_FILE_PATH="/etc/default/grub"

    if ! grep -q "ipv6.disable" "$GRUB_FILE_PATH"; then
      sed -E -e 's/^(GRUB_CMDLINE_LINUX_DEFAULT="[^"]*)"/\1 ipv6.disable=1"/' -i "$GRUB_FILE_PATH"
      update-grub

      bb-flag-set reboot
    fi
```

⚠ After applying the resource, the GRUB settings will be updated and the cluster nodes will begin a sequential reboot to apply the changes.

Additional resources

- Global custom resources
- Module `console` documentation
- Module `control-plane-manager` documentation
- Module `node-manager` custom resources
- Module `operator-trivy` documentation

- Module `user-authz` documentation

Control plane recovery and debugging

Recovery from failures

During its operation DKP automatically creates backups of configuration and data that may be useful in case of problems. These backups are saved in the `/etc/kubernetes/deckhouse/backup` directory. If any issues or unexpected situations occur during operation, you can use these backups to restore the system to a previously healthy state.

Restoring etcd cluster functionality

If the etcd cluster is not functioning and cannot be restored from a backup, you can attempt to recover it from scratch by following the steps below.

1. On all nodes that are part of your etcd cluster, **except one**, delete the `etcd.yaml` manifest located in `/etc/kubernetes/manifests/`. This will leave only one active node, from which the multi-master cluster state will be restored.
2. On the remaining node, open the `etcd.yaml` manifest and add the `--force-new-cluster` flag under `spec.containers.command`.
3. After the cluster is successfully restored, remove the `--force-new-cluster` flag.

✖ This operation is destructive: it completely wipes the existing data and initializes a new cluster based on the state preserved on the remaining node. All pending records will be lost.

Restoring a master node when kubelet fails to load control plane components

Such a situation may occur if images of the control plane components on the master were deleted in a cluster that has a single master node (e.g., the directory `/var/lib/containerd` was deleted). In this case, kubelet cannot pull images of the control plane components when restarted since the master node lacks authorization parameters required for accessing `registry.deckhouse.io`.

Below is an instruction on how you can restore the master node.

containerd

1. Execute the following command to restore the master node in any cluster running under DKP:

```
d8 k -n d8-system get secrets deckhouse-registry -o json |
jq -r '.data.".dockerconfigjson"' | base64 -d |
jq -r '.auths."registry.deckhouse.io".auth'
```

2. Copy the command's output and use it for setting the `AUTH` variable on the corrupted master.
3. Next, pull images of control plane components to the corrupted master:

```
for image in $(grep "image:" /etc/kubernetes/manifests/* | awk '{print $3}'); do
  crictl pull --auth $AUTH $image
done
```

4. Restart kubelet after pulling the images.

etcd restore

Viewing etcd cluster members

Option 1

Use the `etcdctl member list` command.

Example:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

Warning. The last parameter in the output table shows etcd member is in `learner` state, is not in `leader` state.

Option 2

To obtain information about etcd cluster nodes in tabular form, use the `etcdctl endpoint status` command. For the

leader, the `IS LEADER` column will show `true`.

Example:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ endpoint status --cluster -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

Restoring the etcd cluster in case of complete unavailability

1. Stop all etcd nodes except one by deleting the `etcd.yaml` manifest on the others.
2. On the remaining node, add the `--force-new-cluster` option to the etcd startup command.
3. After the cluster is restored, remove this option.

> ✖ Be careful: these actions completely erase the previous data and form a new etcd cluster.

Recovering etcd after panic: unexpected removal of unknown remote peer error

1. Find the `etcdutl` utility on the master node and copy the executable to `/usr/local/bin/`:

   ```
   cp $(find /var/lib/containerd/ \
   -name etcdutl -print -quit) /usr/local/bin/etcdutl
   ```

2. Create a new etcd database snapshot from the current local snapshot (`/var/lib/etcd/member/snap/db`):

   ```
   etcdutl snapshot restore /var/lib/etcd/member/snap/db --name <HOSTNAME> \
   --initial-cluster=<HOSTNAME>=https://<ADDRESS>:2380 --initial-advertise-peer-urls=https://<ADDRESS>:2380 \
   --skip-hash-check=true --data-dir /var/lib/etcdtest
   ```

   where:

   - `<HOSTNAME>`: Name of the master node.
   - `<ADDRESS>`: Address of the master node.

3. Execute the following commands to use the new snapshot:

   ```
   cp -r /var/lib/etcd /tmp/etcd-backup
   rm -rf /var/lib/etcd
   mv /var/lib/etcdtest /var/lib/etcd
   ```

4. Locate the `etcd` and `api-server` containers:

   ```
   crictl ps -a | egrep "etcd|apiserver"
   ```

5. Remove the `etcd` and `api-server` containers:

   ```
   crictl rm <CONTAINER-ID>
   ```

6. Restart the master node.

Actions to take when etcd database exceeds quota-backend-bytes limit

When the database volume of etcd reaches the limit set by the `quota-backend-bytes` parameter, it switches to “read-only” mode. This means that the etcd database stops accepting new entries but remains available for reading data. You can tell that you are facing a similar situation by executing the command:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | sed -n 1p)
-- \
etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ endpoint status -w table --cluster
```

If you see a message like `alarm:NOSPACE` in the `ERRORS` field, you need to take the following steps:

1. Make change to `/etc/kubernetes/manifests/etcd.yaml` — find the line with `--quota-backend-bytes` and increase the value by multiplying the specified number by two. If there is no such line — add, for example: `- --quota-backend-bytes=8589934592` — this sets the limit to 8 GB.
2. Disarm the active alarm that occurred due to reaching the limit. To do this, execute the command:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | sed -n
1p) -- \
etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ alarm disarm
```

3. Change the `maxDbSize` parameter in the `control-plane-manager` settings to match the value specified in the manifest.

etcd defragmentation

⚠ Before defragmenting, back up etcd.

To view the size of the etcd database on a specific node before and after defragmentation, use the command (where `NODE_NAME` is the name of the master node):

```
d8 k -n kube-system exec -it etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  endpoint status --cluster -w table
```

Output example (the size of the etcd database on the node is specified in the `DB SIZE` column):

```
+----------------------------+------------------+---------+-----------------+---------+--------+---
--------------------+--------+------------+------------+-----------+------------+--------------------
-+--------+--------------------------+-------------------+
|          ENDPOINT          |        ID        | VERSION | STORAGE VERSION | DB SIZE | IN USE |
PERCENTAGE NOT IN USE | QUOTA  | IS LEADER  | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED
INDEX | ERRORS | DOWNGRADE TARGET VERSION | DOWNGRADE ENABLED |
+----------------------------+------------------+---------+-----------------+---------+--------+---
--------------------+--------+------------+------------+-----------+------------+--------------------
-+--------+--------------------------+-------------------+
| https://192.168.199.80:2379 | 489a8af1e7acd7a0 |   3.6.1 |           3.6.0 |   76 MB |  62 MB |
20% | 2.1 GB |       true |      false |        56 |  258054684 |          258054684 |        |
|             false |
+----------------------------+------------------+---------+-----------------+---------+--------+---
--------------------+--------+------------+------------+-----------+------------+--------------------
-+--------+--------------------------+-------------------+
| https://192.168.199.81:2379 | 589a8ad1e7ccd7b0 |   3.6.1 |           3.6.0 |   76 MB |  62 MB |
20% | 2.1 GB |      false |      false |        56 |  258054685 |          258054685 |        |
|             false |
+----------------------------+------------------+---------+-----------------+---------+--------+---
--------------------+--------+------------+------------+-----------+------------+--------------------
-+--------+--------------------------+-------------------+
| https://192.168.199.82:2379 | 229a8cd1e7bcd7a0 |   3.6.1 |           3.6.0 |   76 MB |  62 MB |
20% | 2.1 GB |      false |      false |        56 |  258054685 |          258054685 |        |
|             false |
+----------------------------+------------------+---------+-----------------+---------+--------+---
--------------------+--------+------------+------------+-----------+------------+--------------------
-+--------+--------------------------+-------------------+
```

How to defragment an etcd node in a single-master cluster

⚠ Defragmenting etcd is a resource-intensive operation that temporarily blocks etcd from running on that node. Keep this in mind when choosing a time to perform the operation in a cluster with a single master node.

To defragment etcd in a cluster with a single master node, use the following command (where `NODE_NAME` is the name of the master node):

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt \
  --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

Example output when the operation is successful:

```
Finished defragmenting etcd member[https://localhost:2379]. took 848.948927ms
```

> **i** If a timeout error occurs, increase the value of the `–command-timeout` parameter from the command above until defragmentation is successful.

How to defragment etcd in a cluster with multiple master nodes

To defragment etcd in a cluster with multiple master nodes:

1. Get a list of etcd pods. To do this, use the following command:

   ```
   d8 k -n kube-system get pod -l component=etcd -o wide
   ```

   Example output:

   ```
   NAME           READY    STATUS    RESTARTS   AGE     IP              NODE        NOMINATED NODE
   READINESS GATES
   etcd-master-0   1/1     Running   0          3d21h   192.168.199.80  master-0    <none>
   <none>
   etcd-master-1   1/1     Running   0          3d21h   192.168.199.81  master-1    <none>
   <none>
   etcd-master-2   1/1     Running   0          3d21h   192.168.199.82  master-2    <none>
   <none>
   ```

2. Identify the leader master node. To do this, contact any etcd pod and get a list of nodes participating in the etcd cluster using the command (where `NODE_NAME` is the name of the master node):

   ```
   d8 k -n kube-system exec -it etcd-NODE_NAME -- /usr/bin/etcdctl \
     --cert=/etc/kubernetes/pki/etcd/server.crt \
     --key=/etc/kubernetes/pki/etcd/server.key \
     --cacert=/etc/kubernetes/pki/etcd/ca.crt \
     endpoint status --cluster -w table
   ```

   Output example (the leader in the `IS LEADER` column will have the value `true`):

   ```
   +----------------------------+------------------+---------+-----------------+---------+-------
   -+----------------------+--------+------------+------------+-----------+------------+---------
   -----------+--------+--------------------------+-------------------+
   |          ENDPOINT          |        ID        | VERSION | STORAGE VERSION | DB SIZE | IN USE
   | PERCENTAGE NOT IN USE | QUOTA  | IS LEADER  | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT
   APPLIED INDEX | ERRORS | DOWNGRADE TARGET VERSION | DOWNGRADE ENABLED |
   +----------------------------+------------------+---------+-----------------+---------+-------
   -+----------------------+--------+------------+------------+-----------+------------+---------
   -----------+--------+--------------------------+-------------------+
   | https://192.168.199.80:2379 | 489a8af1e7acd7a0 |   3.6.1 |           3.6.0 |   76 MB |  62 MB
   |                  20% | 2.1 GB |       true |      false |        56 |  258054684 |
   258054684 |        |                          |             false |
   +----------------------------+------------------+---------+-----------------+---------+-------
   -+----------------------+--------+------------+------------+-----------+------------+---------
   -----------+--------+--------------------------+-------------------+
   | https://192.168.199.81:2379 | 589a8ad1e7ccd7b0 |   3.6.1 |           3.6.0 |   76 MB |  62 MB
   |                  20% | 2.1 GB |      false |      false |        56 |  258054685 |
   258054685 |        |                          |             false |
   +----------------------------+------------------+---------+-----------------+---------+-------
   -+----------------------+--------+------------+------------+-----------+------------+---------
   -----------+--------+--------------------------+-------------------+
   | https://192.168.199.82:2379 | 229a8cd1e7bcd7a0 |   3.6.1 |           3.6.0 |   76 MB |  62 MB
   |                  20% | 2.1 GB |      false |      false |        56 |  258054685 |
   258054685 |        |                          |             false |
   +----------------------------+------------------+---------+-----------------+---------+-------
   -+----------------------+--------+------------+------------+-----------+------------+---------
   -----------+--------+--------------------------+-------------------+
   ```

3. Defragment the etcd nodes that are members of the etcd cluster one by one. Use the following command to defragment (where `NODE_NAME` is the name of the master node):

   > **i** Important: Defragment the leader last.
   >
   > Restoring etcd on a node after defragmentation may take some time. It is recommended to wait at least a minute before proceeding to defragment the next etcd node.

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt \
  --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

Example output when the operation is successful:

```
Finished defragmenting etcd member[https://localhost:2379]. took 848.948927ms
```

> **i** If a timeout error occurs, increase the value of the `—command-timeout` parameter from the command above until defragmentation is successful.

High availability

If any component of the control plane becomes unavailable, the cluster temporarily maintains its current state but cannot process new events. For example:

- If `kube-controller-manager` fails, Deployment scaling will stop working.
- If `kube-apiserver` is unavailable, no requests can be made to the Kubernetes API, although existing applications will continue to function.

However, prolonged unavailability of control plane components disrupts the processing of new objects, handling of node failures, and other operations. Over time, this can lead to cluster degradation and impact user applications.

To mitigate these risks, the control plane should be scaled to a high-availability configuration — a minimum of three nodes. This is especially critical for etcd, which requires a quorum to elect a leader. The quorum works on a majority basis (N/2 + 1) of the total number of nodes.

Example:

Cluster size	Quorum (majority)	Max fault tolerance
1	1	0
3	2	1
5	3	2
7	4	3
9	5	4

> **i** An even number of nodes does not improve fault tolerance but increases replication overhead.

In most cases, three etcd nodes are sufficient. Use five if high availability is critical. More than seven is rarely necessary and not recommended due to high resource consumption.

After new control plane nodes are added:

- The label `node-role.kubernetes.io/control-plane=""` is applied.
- A DaemonSet launches control plane pods on the new nodes.
- DKP creates or updates files in `/etc/kubernetes`: manifests, configuration files, certificates, etc.
- All DKP modules that support high availability will enable it automatically, unless the global setting `highAvailability` is manually overridden.

Control plane node removal is performed in reverse:

- Labels `node-role.kubernetes.io/control-plane`, `node-role.kubernetes.io/master`, and `node.deckhouse.io/group` are removed.
- DKP removes its pods from these nodes.
- etcd members on the nodes are automatically deleted.
- If the number of nodes drops from two to one, etcd may enter `readonly` mode. In this case, you must start etcd with the `--force-new-cluster` flag, which should be removed after a successful startup.

Additional resources

- Global parameters
- Module `control-plane-manager` parameters

Scaling and changing master nodes

Scaling and single/multi-master transition

Control plane operation modes

Deckhouse Kubernetes Platform (DKP) supports two operation modes for the control plane:

1. **Single-master**:
   - `kube-apiserver` uses only the local `etcd` instance.

    - A proxy server runs on the node to handle requests on `localhost`.
    - The `kube-apiserver` listens only on the master node's IP address.
2. **Multi-master**:
    - `kube-apiserver` interacts with all `etcd` instances in the cluster.
    - A proxy is configured on all nodes:
        - If the local `kube-apiserver` is unavailable, requests are automatically forwarded to other nodes.
    - This ensures high availability and supports scaling.

Automatic scaling of master nodes

DKP allows automatic addition and removal of master nodes using the label `node-role.kubernetes.io/control-plane=""`.

Automatic control of master nodes includes:

- **Adding the label** `node-role.kubernetes.io/control-plane=""` to a node:
    - All control plane components are deployed.
    - The node is added to the etcd cluster.
    - Certificates and configuration files are regenerated automatically.
- **Removing the label**:
    - Control plane components are removed.
    - The node is properly removed from the etcd cluster.
    - Related configuration files are updated.

> **i** Transitioning from 2 to 1 master node requires manual etcd adjustment. All other changes in master node count are handled automatically.

Common scaling scenarios

> ⚠ For cluster stability, it is necessary to maintain an odd number of nodes with etcd to ensure quorum.

DKP supports both automatic and manual scaling of master nodes in cloud and bare-metal clusters:

1. **Single-master → Multi-master**:
    - Add one or more master nodes.
    - Apply the label `node-role.kubernetes.io/control-plane=""` to them.
    - DKP will automatically:
        - Deploy all control plane components.
        - Configure the nodes to work with the `etcd` cluster.
        - Synchronize certificates and configuration files.
2. **Multi-master → Single-master**:
    - Remove the labels `node-role.kubernetes.io/control-plane=""` and `node-role.kubernetes.io/master=""` from the extra master nodes.
    - For **bare-metal clusters**:
        - To correctly remove the nodes from `etcd`:
            - Run `d8 k delete node <node-name>`;
            - Power off the corresponding VMs or servers.

    > **i** **Important**. In cloud clusters, all necessary actions are automatically handled by the `dhctl converge` command.

3. **Changing the number of master nodes in a cloud cluster**:
    - Similar to node addition/removal, typically done using the `dhctl converge` command or cloud tools.
4. **Multi-master migration (3 master nodes) → 2 master nodes and 1 arbiter node in a cloud cluster**:
    - In the cloud provider settings, set `masterNodeGroup.replicas` to `2` and create a NodeGroup for the arbiter node. Similar to node addition/removal, typically done using the `dhctl converge` command or cloud tools.

      For details on configuring HA mode with 2 master nodes and 1 arbiter node in a cloud cluster, refer to Managing HA mode.
5. **Multi-master migration (3 master nodes) → 2 master nodes and 1 arbiter node in a static cluster**:
    - Create a NodeGroup for the arbiter node and add the node to the cluster.
    - Remove the labels `node-role.kubernetes.io/control-plane=""`, `node-role.kubernetes.io/master=""` and `node.deckhouse.io/group-""` from the extra master node.
    - To correctly remove the node from etcd in **bare-metal clusters**:
        - Run `d8 k delete node <node-name>`.
        - Power off the corresponding VM or server.

      For details on configuring HA mode with 2 master nodes and 1 arbiter node in a static cluster, refer to Managing HA mode.

Removing the master role from a node without deleting the node itself

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

If you need to remove a node from the set of master nodes but keep it in the cluster for other purposes, follow these steps:

1. Remove the labels so the node is no longer treated as a master:

```
d8 k label node <node-name> node-role.kubernetes.io/control-plane-
d8 k label node <node-name> node-role.kubernetes.io/master-
d8 k label node <node-name> node.deckhouse.io/group-
```

2. Make sure that the master node to be deleted is no longer listed as a member of the etcd cluster:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

3. Delete the static manifests of the control plane components so they no longer start on the node, and remove unnecessary PKI files. Exec to the node and run the following commands:

```
rm -f /etc/kubernetes/manifests/{etcd,kube-apiserver,kube-scheduler,kube-controller-manager}.yaml
rm -f /etc/kubernetes/{scheduler,controller-manager}.conf
rm -f /etc/kubernetes/authorization-webhook-config.yaml
rm -f /etc/kubernetes/admin.conf /root/.kube/config
rm -rf /etc/kubernetes/deckhouse
rm -rf /etc/kubernetes/pki/{ca.key,apiserver*,etcd/,front-proxy*,sa.*}
rm -rf /var/lib/etcd/member/
```

After completing these steps, the node will no longer be considered a master node, but it will remain part of the cluster and can be used for other tasks.

Changing the OS image of master nodes in a multi-master cluster

The OS change method depends on the cluster type: in a cloud cluster, nodes are replaced with `dhctl converge`; in a static cluster, nodes are replaced manually, one at a time.

In a cloud cluster

To change the OS image of master nodes in a cloud multi-master cluster, follow these steps:

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

1. Create a backup of etcd and the `/etc/kubernetes` directory.
2. Copy the resulting archive outside the cluster (e.g., to a local machine).
3. Make sure there are no alerts in the cluster that could interfere with updating master nodes.
4. Ensure the DKP queue is empty:

```
d8 system queue list
```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

```
docker login registry.deckhouse.io
```

   During the login process, you will need to enter your `Username` and `Password`.

   i When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Kubernetes Platform license key.

6. **On your local machine**, run the Deckhouse installer container for the corresponding edition and version (change the container registry address if necessary):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
  registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. **In the installer container**, run the following command to check the state before starting the operation:

   ```
   dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
     --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
   ```

   The output should indicate that Terraform has found no discrepancies and no changes are required.

8. **In the installer container**, run the following command and specify the desired OS image in the `masterNodeGroup.instanceClass` parameter
   (provide all master node addresses using the `--ssh-host` parameter):

   ```
   dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-
   user=<USERNAME> \
     --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
   ```

9. **In the installer container**, run the following command to update the nodes:

   Carefully review the actions that `converge` plans to perform when it prompts for confirmation.

   During execution, nodes will be replaced with new ones, one by one, starting from the highest numbered node (2) down to the lowest (0):

   ```
   dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
     --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
   ```

   The following steps should be performed **sequentially on each** master node, starting with the highest numbered node (with suffix 2) and ending with the lowest (with suffix 0).

10. **On the newly created node**, open the systemd journal for the `bashible.service`.
    Wait until the setup process is complete — the log should contain the message `nothing to do`:

    ```
    journalctl -fu bashible.service
    ```

11. Verify that the etcd node appears in the cluster node list:

    ```
    for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
      d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
      --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
      --endpoints https://127.0.0.1:2379/ member list -w table
      if [ $? -eq 0 ]; then
        break
      fi
    done
    ```

12. Make sure that `control-plane-manager` is running on the node:

    ```
    d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
      -l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER-NODE-N-NAME>
    ```

13. Check the Deckhouse queue and make sure that there are no errors with the command:

    ```
    d8 system queue list
    ```

14. Proceed to updating the next master node.

In a static cluster

The following instruction describes OS replacement on master nodes that were added to the cluster **manually** using the `bootstrap.sh` script. Perform the steps for each master node, one node at a time. Do not proceed to the next node until the current node has rejoined the cluster and is healthy.

> ⚠ If master nodes are managed by Cluster API Provider Static (CAPS) through StaticInstance resources, do not use this instruction. First delete the StaticInstance, install the required OS, and add the node to the `master` NodeGroup again.

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

To change the OS of a manually added master node, follow these steps:

1. Create a backup of etcd and the `/etc/kubernetes` directory. If the `stronghold` module is enabled, make sure its data has also been backed up.
2. Check the cluster health and make sure there are no alerts or pending Deckhouse queue tasks:

   ```
   d8 status
   ```

3. Remove the master node labels from the node:

   ```
   d8 k label node <MASTER_NODE_NAME> node-role.kubernetes.io/control-plane- \
     node-role.kubernetes.io/master- node.deckhouse.io/group-
   ```

   where `<MASTER_NODE_NAME>`: Name of the master node being changed.
4. Make sure the node has been removed from the etcd cluster member list:

   ```
   for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
     d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
       --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
       --endpoints https://127.0.0.1:2379/ member list -w table
    if [ $? -eq 0 ]; then
      break
    fi
   done
   ```

5. Evict workloads from the node with `d8 k drain`:

   ```
   d8 k drain <MASTER_NODE_NAME> --ignore-daemonsets --delete-emptydir-data
   ```

6. Force-delete any Pods remaining on the node:

   ```
   d8 k delete pods --all-namespaces --field-selector spec.nodeName=<MASTER_NODE_NAME> --force
   ```

7. Delete the Node object:

   ```
   d8 k delete node <MASTER_NODE_NAME>
   ```

8. Clean up the DKP data on the removed master node. This command irreversibly removes Kubernetes and DKP data from the node. Before running it, make sure you have selected the correct node and created the required backups:

   ```
   bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
   ```

9. Install the required OS on the node.
10. On one of the remaining master nodes in the cluster, retrieve and decode the script for adding a master node:

    ```
    d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-master \
      -o jsonpath='{.data.bootstrap\.sh}' | base64 -d > bootstrap.sh
    ```

11. Securely copy the `bootstrap.sh` file obtained in the previous step to the node being added and run it as `root` on that node:

    ```
    bash bootstrap.sh
    ```

12. **On the node being added**, open the systemd journal for the `bashible.service`. Wait until the setup process is complete — the log should contain the message `nothing to do`:

    ```
    journalctl -fu bashible.service
    ```

13. Wait until the node reaches the `Ready` status:

    ```
    d8 k wait node <MASTER_NODE_NAME> --for=condition=Ready --timeout=10m
    ```

14. Make sure the node is listed as an etcd cluster member:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
   --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
   --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

15. Make sure that `control-plane-manager` is running on the node:

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
  -l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER_NODE_NAME>
```

16. Make sure there are no alerts or pending queue tasks:

```
d8 status
```

17. Repeat the procedure for the next master node.

Changing the OS image in a single-master cluster

The method depends on the cluster type: first add extra master nodes, replace the OS in multi-master mode, then restore the original number of master nodes.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

In a cloud cluster

1. Convert the single-master cluster into a multi-master one according to the instructions.
2. Change the OS on the master nodes according to the instructions.
3. Convert the multi-master cluster back to a single-master one following the instructions.

In a static cluster

1. Add extra master nodes according to the instructions.
2. Change the OS on the master nodes according to the instructions.
3. Remove extra master nodes from the control plane role according to the instructions. Then delete them from the cluster with `d8 k delete node <MASTER_NODE_NAME>` and power off the corresponding servers.

Adding master nodes to a static or hybrid cluster

i It is important to have an odd number of masters to ensure a quorum.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

When installing Deckhouse Kubernetes Platform with default settings, the NodeGroup `master` lacks the section `spec.staticInstances.labelSelector` with label filter settings for `staticInstances` resources. Because of this, after changing the number of `staticInstances` nodes in the NodeGroup `master` (parameter `spec.staticInstances.count`), when adding a regular node using Cluster API Provider Static (CAPS), it can be "intercepted" and added to the NodeGroup `master`, even if the corresponding `StaticInstance` (in `metadata`) specifies a label with a `role` different from `master`. To avoid this "interception", after installing DKP, edit the NodeGroup `master` — add the section `spec.staticInstances.labelSelector` with label filter settings for `staticInstances` resources. Example of NodeGroup `master` with `spec.staticInstances.labelSelector`:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: master
spec:
  nodeType: Static
  staticInstances:
    count: 2
    labelSelector:
      matchLabels:
        role: master
```

Next, when adding master nodes to the cluster using CAPS, specify the label specified in `spec.staticInstances.labelSelector` NodeGroup `master` in the corresponding `StaticInstance`. Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-master-1
  labels:
    # The label specified in spec.staticInstances.labelSelector NodeGroup master.
    role: master
spec:
  # Specify the IP address of the static node server.
  address: "<SERVER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
```

i When adding new master nodes using CAPS and changing the number of master nodes in the NodeGroup `master` (parameter `spec.staticInstances.count`), please note the following:

During the initial cluster installation, the configuration specifies the first master node on which the installation takes place. If, after installation, you need to create a multi-master cluster and add master nodes using CAPS, you must specify the number of nodes in the `spec.staticInstances.count` parameter of the NodeGroup `master` as one less than the desired number.

For example, if you need to create a multi-master with three master nodes in `spec.staticInstances.count` NodeGroup `master`, specify the value `2` and create two `staticInstances` for the nodes to be added. After adding them to the cluster, the number of master nodes will be three: the master node on which the installation took place and two master nodes added using CAPS.

Otherwise, adding a master node to a static or hybrid cluster is similar to adding a regular node. To do this, use the corresponding examples. All the necessary actions to configure a cluster control plane components on the new master nodes are performed automatically. Wait until the master nodes appear in `Ready` status.

Adding master nodes in a cloud cluster

This section describes how to convert a single-master cluster into a multi-master cluster.

⚠ Before adding nodes, make sure the required quotas are available. It's important to have an odd number of master nodes to maintain etcd quorum.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

1. Create a backup of etcd and the `/etc/kubernetes` directory.
2. Copy the resulting archive outside the cluster (e.g., to a local machine).
3. Ensure there are no active alerts in the cluster that may interfere with adding new master nodes.
4. Make sure the Deckhouse queue is empty:

   ```
   d8 system queue list
   ```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

   ```
   docker login registry.deckhouse.io
   ```

   During the login process, you will need to enter your `Username` and `Password`.

   i When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Kubernetes Platform license key.

6. On the **local machine**, run the Deckhouse installer container for the appropriate edition and version (change the container registry address if necessary):

   ```
   DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
   DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]' )
   docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
     registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
   ```

7. **In the installer container**, run the following command to verify the state before proceeding:

```
dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host
<MASTER-NODE-0-HOST>
```

The output should confirm that Terraform found no differences and no changes are needed.

8. **In the installer container**, run the following command and set the target number of master nodes in the `masterNodeGroup.replicas` parameter:

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-
user=<USERNAME> \
  --ssh-host <MASTER-NODE-0-HOST>
```

> **i** For **Yandex Cloud**, if public IPs are assigned to master nodes, the number of elements in the `masterNodeGroup.instanceClass.externalIPAddresses` array must match the number of master nodes. Even when using the `Auto` value (automatic public IP assignment), the number of items in the array must still match.
>
> For example, with three master nodes (`masterNodeGroup.replicas: 3`) and automatic IP assignment, the `masterNodeGroup.instanceClass.externalIPAddresses` section would look like:
>
> ```
> externalIPAddresses:
> - "Auto"
> - "Auto"
> - "Auto"
> ```

9. **In the installer container**, run the following command to trigger scaling:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host
<MASTER-NODE-0-HOST>
```

10. Wait until the required number of master nodes reaches the `Ready` status and all `control-plane-manager` pods become ready:

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady -l app=d8-control-plane-manager
```

Reducing the number of master nodes in a cloud cluster

This section describes the process of converting a multi-master cluster into a single-master cluster.

> ⚠ The following steps must be performed starting from the first master node (`master-0`) in the cluster. This is because the cluster scales in order — for example, it is not possible to remove `master-0` and `master-1` while leaving `master-2` .

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

1. Create a backup of etcd and the `/etc/kubernetes` directory.
2. Copy the resulting archive outside the cluster (e.g., to a local machine).
3. Ensure there are no alerts in the cluster that may interfere with the master node update process.
4. Make sure the DKP queue is empty:

```
d8 system queue list
```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

```
docker login registry.deckhouse.io
```

During the login process, you will need to enter your `Username` and `Password` .

> **i** When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token` , and the `Password` field must contain the Deckhouse Kubernetes Platform license key.

6. On the **local machine**, run the DKP installer container for the corresponding edition and version (change the container registry address if needed):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]' '[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
  registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. **In the installer container**, run the following command and set `masterNodeGroup.replicas` to `1`:

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> \
  --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
```

> **i** For **Yandex Cloud**, if external IPs are used for master nodes, the number of items in the `masterNodeGroup.instanceClass.externalIPAddresses` array must match the number of master nodes. Even when using `Auto` (automatic public IP allocation), the number of entries must still match.
>
> For example, for a single master node (`masterNodeGroup.replicas: 1`) and automatic IP assignment, the `masterNodeGroup.instanceClass.externalIPAddresses` section would look like:
>
> ```
> externalIPAddresses:
> - "Auto"
> ```

8. **In the installer container**, run the following command to trigger the scaling operation:

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host
<MASTER-NODE-0-HOST>
```

> **i** For **OpenStack** and **VK Cloud (OpenStack)**, after confirming the node deletion, it is extremely important to check the disk deletion `<prefix>kubernetes-data-N` in Openstack itself.
>
> For example, when deleting the `cloud-demo-master-2` node in the Openstack web interface or in the OpenStack CLI, it is necessary to check the absence of the `cloud-demo-kubernetes-data-2` disk.
>
> If the kubernetes-data disk remains, there may be problems with ETCD operation as the number of master nodes increases.

9. Check the Deckhouse queue and make sure that there are no errors with the command:

```
d8 system queue list
```

Accessing the DKP controller in a multi-master cluster

In clusters with multiple master nodes, DKP runs in high-availability mode (with multiple replicas). To access the active DKP controller, you can use the following command (example shown for the `deckhouse-controller queue list` command):

```
d8 system queue list
```

Additional resources

- Module `control-plane-manager` documentation
- Module `node-manager` custom resources
- Module `stronghold` documentation

Node management basics in Deckhouse

Deckhouse Kubernetes Platform (DKP) supports the full lifecycle of node management:

- Automatic node scaling based on workload.
- Node updates and maintenance to keep them up to date.
- Centralized configuration management for node groups using the NodeGroup CRD.
- Support for various types of nodes: permanent, ephemeral, cloud-based, or bare-metal.

> **i** DKP can operate on both bare-metal and cloud-based clusters, providing flexibility and scalability.

Node groups allow logical segmentation of the cluster infrastructure. In DKP, the following NodeGroup roles are commonly used:

- `master`: Control plane nodes.
- `front`: Nodes for routing HTTP(S) traffic.
- `monitoring`: Nodes for hosting monitoring components.
- `worker`: Nodes for user applications.

- `system`: Dedicated nodes for system components.

Each group can have centralized configuration settings, including the Kubernetes version, resources, taints, labels, kubelet parameters, and more.

> **i** Node management is only available for clusters fully managed by DKP. This feature is not supported for Managed Kubernetes clusters.

Automatic deployment and updates

Deckhouse Kubernetes Platform (DKP) provides an automated mechanism for managing the lifecycle of nodes based on NodeGroup resources. DKP supports both initial node provisioning and updates when configuration changes, for both cloud and bare-metal clusters (if the `node-manager` module is enabled).

How it works:

1. A NodeGroup is the primary resource for managing groups of nodes. It defines the node type, number of nodes, resource templates, and key parameters (e.g., kubelet settings, taints, etc.).
2. When a NodeGroup is created or modified, the `node-manager` module automatically reconciles the node state with the specified configuration.
3. Updates occur without user intervention — outdated nodes are removed, and new ones are created automatically.

Let’s take a look at automatic updates using the example of a kubelet version upgrade.

1. The user updates the `kubelet` section in the NodeGroup specification.
2. DKP detects that current nodes do not match the new configuration.
3. New nodes with the updated settings are created sequentially.
4. Old nodes are gradually removed from the cluster.

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-cloud
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: AnotherCloudInstanceClass
      name: my-class
```

Basic node and OS configuration

When nodes are created and joined to the cluster, DKP automatically performs a series of actions required for proper cluster operation:

- Installing and configuring a supported operating system.
- Disabling automatic package updates.
- Setting up logging and system parameters.
- Installing necessary packages and utilities.
- Configuring the `kubernetes-api-proxy` component to balance traffic between `kubelet` and API servers.
- Installing and configuring the container runtime (`containerd`) and `kubelet`.
- Registering the node with the Kubernetes cluster.

These actions are performed automatically when using `bootstrap.sh` or when connecting nodes via StaticInstance and SSHCredentials resources.

Disruptive updates

Some updates — for example, upgrading `containerd` or kubelet across multiple versions — require node downtime and may cause short-term disruption of system components (*disruptive updates*).
The application mode for such updates is configured via the `disruptions.approvalMode` parameter:

- `Manual`: Manual approval mode for disruptive updates.
  When a disruptive update is available, the `NodeRequiresDisruptionApprovalForUpdate` alert is triggered.

  To approve the update, add the annotation `update.node.deckhouse.io/disruption-approved=` to each node in the group. Example:

  ```
  d8 k annotate node <NODE-NAME> update.node.deckhouse.io/disruption-approved=
  ```

  > **i** **Important**: In this mode, the node is not drained automatically.
  > If needed, perform the drain manually before applying the annotation.
  >
  > To avoid issues during draining, always use the `Manual` mode for master node groups.

- `Automatic`: Automatic approval mode for disruptive updates.

  In this mode, the node is drained automatically before applying the update by default.

This behavior can be changed using the `disruptions.automatic.drainBeforeApproval` parameter in the node group settings.

- `RollingUpdate`: A mode in which a new node with updated settings is created and the old one is removed. Applicable only to cloud nodes.

  In this mode, an additional node is created in the cluster during the update.
  This can be useful if the cluster lacks sufficient resources to temporarily relocate workloads from the updating node.

Example of a system NodeGroup

System nodes are dedicated to running system components.
They are typically isolated using labels and taints to prevent user pods from being scheduled on them.
System nodes can be either static or cloud-based.

Example:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: system
spec:
  nodeTemplate:
    labels:
      node-role.deckhouse.io/system: ""
    taints:
      - effect: NoExecute
        key: dedicated.deckhouse.io
        value: system
  nodeType: Static
```

Example NodeGroupConfiguration descriptions

Installing the cert-manager plugin for kubectl on master nodes

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-cert-manager-plugin.sh
spec:
  weight: 100
  bundles:
  - "*"
  nodeGroups:
  - "master"
  content: |
    if [ -x /usr/local/bin/kubectl-cert_manager ]; then
      exit 0
    fi
    curl -L https://github.com/cert-manager/cert-manager/releases/download/v1.7.1/kubectl-
cert_manager-linux-amd64.tar.gz -o - | tar -zxvf - kubectl-cert_manager
    mv kubectl-cert_manager /usr/local/bin
```

Werf configuration for ignoring the Ready status of a node group

Werf checks the `Ready` status of resources and, if available, waits for the value to become `True`.

Creating (or updating) a NodeGroup resource in the cluster may take a significant amount of time (until all nodes become ready). When using werf (e.g., in CI/CD), this can lead to a build timeout.

To make werf ignore the NodeGroup status, add the following annotations to the NodeGroup resource:

```
metadata:
  annotations:
    werf.io/fail-mode: IgnoreAndContinueDeployProcess
    werf.io/track-termination-mode: NonBlocking
```

Settings for Static and CloudStatic NodeGroups

Node groups with types Static and CloudStatic are intended for managing manually created nodes — either physical (bare-metal) or virtual (in the cloud, but outside DKP automation). These nodes are connected manually or via StaticInstance and do not support automatic updates or scaling.

Configuration specifics:

- All update operations (e.g., kubelet updates, node restarts, replacements) must be performed manually or through

external automation tools outside of DKP.

- It is recommended to explicitly set the desired kubelet version to ensure consistency across nodes, especially if they are added with different versions manually:

```
nodeTemplate:
  kubelet:
    version: "1.28"
```

- Node registration to the cluster can be performed either manually or automatically, depending on the configuration:
  - **Manual**: The user downloads the bootstrap script, configures the server, and runs the script manually.
  - **Automatic (CAPS)**: When using StaticInstance and SSHCredentials, DKP automatically connects and configures the nodes.
  - **Hybrid approach**: A manually added node can be handed over to CAPS by using the annotation `static.node.deckhouse.io/skip-bootstrap-phase: ""`.

If the Cluster API Provider Static (CAPS) is enabled, the NodeGroup resource can use the `staticInstances` section. This allows DKP to automatically connect, configure, and, if necessary, clean up static nodes based on StaticInstance and SSHCredentials resources.

> **i** In a NodeGroup of type Static or CloudStatic, you can explicitly specify the number of nodes using the `spec.staticInstances.count` parameter. This allows you to define the expected number of nodes — DKP uses this value for state monitoring and automation.

Running DKP on an arbitrary node

To run DKP on an arbitrary node, configure the `deckhouse` module with the appropriate `nodeSelector` parameter and **do not** specify `tolerations`. The required `tolerations` will be set automatically in this case.

> ⚠ Only use nodes of type **CloudStatic** or **Static** to run DKP. Avoid using a `NodeGroup` that contains only a single node for running DKP.

Example module configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    nodeSelector:
      node-role.deckhouse.io/deckhouse: ""
```

Additional resources

- Module `deckhouse` parameters
- Module `node-manager` custom resources

Adding and managing cloud nodes

In Deckhouse Kubernetes Platform (DKP), cloud nodes can be of the following types:

- **CloudEphemeral** — temporary nodes that are automatically created and deleted.
- **CloudPermanent** — permanent nodes managed manually via `replicas`.
- **CloudStatic** — static cloud nodes. The machines are created manually or by external tools and DKP connects them to the cluster and manages them just like the regular nodes.

Below are instructions for adding and configuring each type.

Adding CloudEphemeral nodes in a cloud cluster

CloudEphemeral nodes are automatically created and managed within the cluster using the Machine Controller Manager (MCM) or Cluster API (depending on configuration) — both components are part of the `node-manager` module in DKP.

To add nodes:

1. Ensure that the cloud provider module is enabled. For example: `cloud-provider-yandex`, `cloud-provider-openstack`, `cloud-provider-aws`.
2. Create an InstanceClass object with the machine configuration. This object describes the parameters of the virtual machines to be created in the cloud.

   Example (for OpenStack):

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: worker-instance
spec:
  flavorName: m1.medium
  imageName: ubuntu-24-04-cloud-amd64
  rootDiskSize: 20
  mainNetwork: default
```

The following parameters are specified:

- `flavorName`: Instance type (CPU/RAM).
- `imageName`: OS image.
- `rootDiskSize`: Size of the root disk.
- `mainNetwork`: Cloud network for the instance.

3. Create a NodeGroup with the `CloudEphemeral` type. Example manifest:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: workers
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: OpenStackInstanceClass
      name: worker-instance
    minPerZone: 1
    maxPerZone: 3
    zones:
      - nova
  nodeTemplate:
    labels:
      node-role.deckhouse.io/worker: ""
    taints: []
```

4. Wait for the nodes to be launched and added automatically.

Configuration for CloudEphemeral NodeGroups

NodeGroups of CloudEphemeral type are designed for automatic scaling by creating and removing virtual machines in the cloud using the Machine Controller Manager (MCM). This type of groups is commonly used in cloud-based DKP clusters.

Node configuration is defined in the `cloudInstances` section and includes parameters for scaling, zoning, fault tolerance, and prioritization.

Example basic configuration:

```
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    minPerZone: 1
    maxPerZone: 5
    maxUnavailablePerZone: 1
    zones:
    - ru-central1-a
    - ru-central1-b
```

Modifying the cloud provider configuration in a cluster

The configuration of the cloud provider used in a cloud or hybrid cluster is stored in the `<PROVIDER_NAME>ClusterConfiguration` structure, where `<PROVIDER_NAME>` is the name/code of the provider. For example, for the OpenStack provider, the structure is called OpenStackClusterConfiguration.

Regardless of the cloud provider used, its settings can be modified using the following command:

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

NodeGroup autoscaling

In Deckhouse Kubernetes Platform (DKP), node group autoscaling is performed based on resource demands (CPU and memory) by the `Cluster Autoscaler` component, which is part of the `node-manager` module.

Autoscaling is triggered only when there are Pending pods that cannot be scheduled on existing nodes due to insufficient

resources (e.g., CPU or memory). In this case, `Cluster Autoscaler` attempts to add nodes based on the NodeGroup configuration.

Key scaling parameters are defined in the `cloudInstances` section of the NodeGroup resource:

- `minPerZone`: The minimum number of virtual machines per zone. This number is always maintained, even with no workload.
- `maxPerZone`: The maximum number of nodes that can be created per zone. This defines the upper scaling limit.
- `maxUnavailablePerZone`: Limits the number of unavailable nodes during updates, deletions, or creation.
- `standby`: Optional parameter that allows pre-provisioning standby nodes.
- `priority`: Integer priority of the group. When scaling, `Cluster Autoscaler` prefers groups with a higher `priority` value. This allows you to define scaling order among multiple NodeGroups. You can control the rate at which nodes are switched to lower-priority groups on demand (for more details, see the FAQ).

Example configuration of a NodeGroup with autoscaling:

```
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    minPerZone: 1       # Minimum number of nodes per zone.
    maxPerZone: 5       # Maximum number of nodes per zone.
    maxUnavailablePerZone: 1  # Number of nodes that can be updated/removed simultaneously.
    zones:
      - nova
      - supernova
      - hypernova
```

Example autoscaling scenario

Assume the following NodeGroup configuration:

```
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: OpenStackInstanceClass
      name: m4.large
    minPerZone: 1
    maxPerZone: 5
    zones:
      - nova
      - supernova
      - hypernova
```

There is also a Deployment with the following configuration:

```
kind: Deployment
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: app
        resources:
          requests:
            cpu: 1500m
            memory: 5Gi
```

In this case, launching all replicas will require 3 nodes — one in each zone.

Now let’s increase the number of replicas to 5. As a result, two Pods will enter the `Pending` state.

`Cluster Autoscaler`:

- Detects the situation.
- Calculates the amount of lacking resources.
- Decides to create two more nodes.
- Hands off the task to the Machine Controller Manager.
- Two new VMs will be created in the cloud and automatically join the cluster.
- The Pods will be scheduled onto the new nodes.

Allocating nodes for specific workloads

⚠ You cannot use the `deckhouse.io` domain in `labels` and `taints` keys of the NodeGroup. It is reserved for DKP components. Use the `dedicated` or `dedicated.client.com` keys instead.

There are two ways to solve this problem:

1. You can set labels to NodeGroup `spec.nodeTemplate.labels`, to use them in the `Pod`'s `spec.nodeSelector` or `spec.affinity.nodeAffinity` parameters. In this case, you select nodes that the scheduler will use for running the target application.
2. You cat set taints to NodeGroup's `spec.nodeTemplate.taints` and then remove them via the `Pod`'s `spec.tolerations` parameter. In this case, you disallow running applications on these nodes unless those applications are explicitly allowed.

> **i**
>
> DKP tolerates the `dedicated` key by default, so we recommend using the `dedicated` key with any value for taints on your dedicated nodes.
>
> To use custom keys for taints (e.g., `dedicated.client.com`), you must add the key's value to the `modules.placement.customTolerationKeys` parameter. This way, Deckhouse can deploy system components (e.g., `cni-flannel`) to these dedicated nodes.

Accelerating node provisioning in the cloud during horizontal application scaling

To speed up the launch of new application replicas during automatic horizontal scaling, it is recommended to maintain a certain number of pre-provisioned (standby) nodes in the cluster.
This allows new application Pods to be scheduled quickly without waiting for node creation and initialization. Keep in mind that having standby nodes increases infrastructure costs.

The target NodeGroup configuration should be as follows:

1. Specify the absolute number of pre-provisioned nodes (or a percentage of the maximum number of nodes in the group) using the `cloudInstances.standby` parameter.
2. If there are additional service components on nodes that are not handled by Deckhouse (e.g., the `filebeat` DaemonSet), you can specify the percentage of node resources they can consume via the `standbyHolder.overprovisioningRate` parameter.
3. This feature requires that at least one group node is already running in the cluster. In other words, there must be either a single replica of the application, or the `cloudInstances.minPerZone` parameter must be set to `1`.

Example:

```
cloudInstances:
  maxPerZone: 10
  minPerZone: 1
  standby: 10%
  standbyHolder:
    overprovisioningRate: 30%
```

Example NodeGroup definition

Cloud nodes

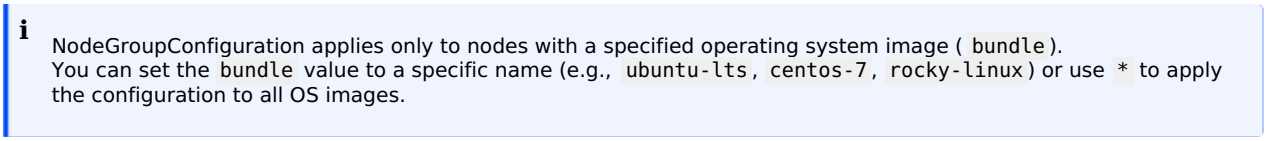
```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: test
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    zones:
      - eu-west-1a
      - eu-west-1b
    minPerZone: 1
    maxPerZone: 2
    classReference:
      kind: AWSInstanceClass
      name: test
  nodeTemplate:
    labels:
      tier: test
```

Configuring CloudEphemeral nodes using NodeGroupConfiguration

Additional settings for cloud nodes can be defined using NodeGroupConfiguration objects. These allow you to:

- Modify OS parameters (e.g., `sysctl`).
- Add root certificates.
- Configure trust for private image registries, etc.

NodeGroupConfiguration is applied to new nodes at creation time, including CloudEphemeral nodes.

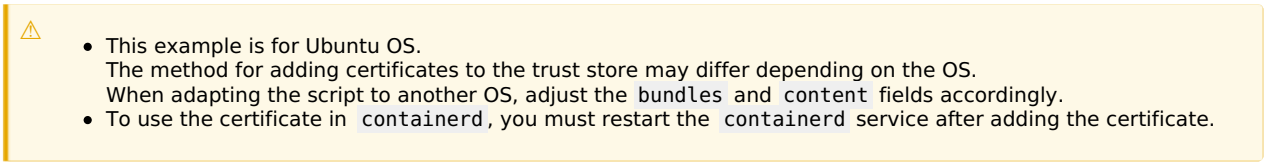


Example NodeGroupConfiguration definitions

Setting a sysctl parameter

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: sysctl-tune.sh
spec:
  weight: 100
  bundles:
  - "*"
  nodeGroups:
  - "*"
  content: |
    sysctl -w vm.max_map_count=262144
```

Adding a root certificate to the host



```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-custom-ca.sh
spec:
  weight: 31
  nodeGroups:
  - '*'
  bundles:
  - 'ubuntu-lts'
  content: |-
    CERT_FILE_NAME=example_ca
    CERTS_FOLDER="/usr/local/share/ca-certificates"
    CERT_CONTENT=$(cat <<EOF
    -----BEGIN CERTIFICATE-----
    MIIDSjCCAjKgAwIBAgIRAJ4RR/WDuAym7M11JA8W7D0wDQYJKoZIhvcNAQELBQAw
    ...
    -----END CERTIFICATE-----
    EOF
    )

    # bb-event           - Creating subscription for event function. More information:
http://www.bashbooster.net/#event
    ## ca-file-updated   - Event name
    ## update-certs      - The function name that the event will call

    bb-event-on "ca-file-updated" "update-certs"

    update-certs() {          # Function with commands for adding a certificate to the store
      update-ca-certificates
    }

    # bb-tmp-file - Creating temp file function. More information: http://www.bashbooster.net/#tmp
    CERT_TMP_FILE="$( bb-tmp-file )"
    echo -e "${CERT_CONTENT}" > "${CERT_TMP_FILE}"

    # bb-sync-file                                - File synchronization function. More information:
http://www.bashbooster.net/#sync
    ## "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"    - Destination file
    ##  ${CERT_TMP_FILE}                          - Source file
    ##  ca-file-updated                           - Name of event that will be called if the file
changes.

    bb-sync-file \
      "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" \
      ${CERT_TMP_FILE} \
      ca-file-updated
```

Adding a certificate to the OS and containerd

> ⚠ This example is for Ubuntu OS.
> The method for adding certificates to the trust store may differ depending on the OS.
> When adapting the script to another OS, update the `bundles` and `content` parameters accordingly.

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-custom-ca-containerd.sh
spec:
  weight: 31
  nodeGroups:
  - '*'
  bundles:
  - 'ubuntu-lts'
  content: |-
    REGISTRY_URL=private.registry.example
    CERT_FILE_NAME=${REGISTRY_URL}
    CERTS_FOLDER="/usr/local/share/ca-certificates"
    CERT_CONTENT=$(cat <<EOF
    -----BEGIN CERTIFICATE-----
    MIIDSjCCAjKgAwIBAgIRAJ4RR/WDuAym7M11JA8W7D0wDQYJKoZIhvcNAQELBQAw
    ...
    -----END CERTIFICATE-----
    EOF
    )
    CONFIG_CONTENT=$(cat <<EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
        ca_file = "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"
    EOF
    )

    mkdir -p /etc/containerd/conf.d

    # bb-tmp-file - Create temp file function. More information: http://www.bashbooster.net/#tmp

    CERT_TMP_FILE="$( bb-tmp-file )"
    echo -e "${CERT_CONTENT}" > "${CERT_TMP_FILE}"

    CONFIG_TMP_FILE="$( bb-tmp-file )"
    echo -e "${CONFIG_CONTENT}" > "${CONFIG_TMP_FILE}"

    # bb-event           - Creating subscription for event function. More information:
http://www.bashbooster.net/#event
    ## ca-file-updated   - Event name
    ## update-certs      - The function name that the event will call

    bb-event-on "ca-file-updated" "update-certs"

    update-certs() {          # Function with commands for adding a certificate to the store
      update-ca-certificates  # Restarting the containerd service is not required as this is done
automatically in the script 032_configure_containerd.sh
    }

    # bb-sync-file                                - File synchronization function. More information:
http://www.bashbooster.net/#sync
    ## "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"    - Destination file
    ##  ${CERT_TMP_FILE}                          - Source file
    ##  ca-file-updated                           - Name of event that will be called if the file
changes.

    bb-sync-file \
      "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" \
      ${CERT_TMP_FILE} \
      ca-file-updated

    bb-sync-file \
      "/etc/containerd/conf.d/${REGISTRY_URL}.toml" \
      ${CONFIG_TMP_FILE}
```

Kernel update on nodes

Debian-based distros

Create a NodeGroupConfiguration resource by specifying the target kernel version in the `desired_version` variable of the shell script (the resource's `spec.content` parameter):

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-kernel.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 32
  content: |
    # Copyright 2022 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    desired_version="5.15.0-53-generic"

    bb-event-on 'bb-package-installed' 'post-install'
    post-install() {
      bb-log-info "Setting reboot flag due to kernel was updated"
      bb-flag-set reboot
    }

    version_in_use="$(uname -r)"

    if [[ "$version_in_use" == "$desired_version" ]]; then
      exit 0
    fi

    bb-deckhouse-get-disruptive-update-approval
    bb-apt-install "linux-image-${desired_version}"
```

CentOS-based distros

Create a NodeGroupConfiguration resource by specifying the target kernel version in the `desired_version` variable of the shell script (the resource's `spec.content` parameter):

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-kernel.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 32
  content: |
    # Copyright 2022 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    desired_version="3.10.0-1160.42.2.el7.x86_64"

    bb-event-on 'bb-package-installed' 'post-install'
    post-install() {
      bb-log-info "Setting reboot flag due to kernel was updated"
      bb-flag-set reboot
    }

    version_in_use="$(uname -r)"

    if [[ "$version_in_use" == "$desired_version" ]]; then
      exit 0
    fi

    bb-deckhouse-get-disruptive-update-approval
    bb-dnf-install "kernel-${desired_version}"
```

Adding CloudPermanent nodes to a cloud cluster

To add CloudPermanent nodes to a DKP cloud cluster:

1. Make sure the cloud provider module is enabled. For example: `cloud-provider-yandex`, `cloud-provider-openstack`, `cloud-provider-aws`, etc.

   You can check this by running the following command:

   ```
   d8 k -n d8-system get modules
   ```

   Or view it in the Deckhouse web interface.

2. Create a NodeGroup object with the `CloudPermanent` type. Nodes of this type are managed via Terraform, which is integrated into DKP. The configuration for such nodes is located in the `(Provider)ClusterConfiguration` object. You edit this configuration using the `dhctl` utility from the installation container. Example:

   ```
   nodeGroups:
   - name: cloud-permanent
     replicas: 2
     instanceClass:
       flavorName: m1.large
       imageName: ubuntu-24-04-cloud-amd64
       rootDiskSize: 20
       mainNetwork: default
     volumeTypeMap:
       nova: ceph-ssd
   ```

3. Specify the instance template parameters. The fields inside the `instanceClass` section depend on the specific cloud provider. Below is an example for OpenStack:
   - `flavorName`: Instance type (resources: CPU, RAM).
   - `imageName`: OS image.
   - `rootDiskSize`: Size of the root disk (in GB).
   - `mainNetwork`: Network name.

- If needed: etcd disk, zones, volume types, etc.

For other cloud providers, the field names and structure may differ. For specific fields, refer to the CRD definition or the documentation for the corresponding cloud provider.

4. Apply the configuration using `dhctl converge`. After editing the `(Provider)ClusterConfiguration`, run:

   ```
   dhctl converge \
     --ssh-host <master node IP> \
     --ssh-user <username> \
     --ssh-agent-private-keys /tmp/.ssh/<key>
   ```

   This command will:

   - Launch Terraform.
   - Create the required virtual machines.
   - Install DKP on them (using `bootstrap.sh`).
   - Register the nodes in the cluster.

5. Done — the new nodes will automatically appear in the cluster.
   You can view them by running:

   ```
   d8 k get nodes
   ```

   The list of newly created nodes is also available in the Deckhouse web interface.

Deckhouse Kubernetes Platform can run on top of Managed Kubernetes services (e.g., GKE and EKS).
In such cases, the `node-manager` module provides node configuration management and automation,
but its capabilities may be limited by the respective cloud provider's API.

Adding a CloudStatic node to a cluster

Adding a static node can be done manually or using the Cluster API Provider Static (CAPS).

Manually

Follow the steps below to add a new static node (e.g., VM or bare metal server) to the cluster:

1. For CloudStatic nodes in the following cloud providers, refer to the steps outlined in the documentation:
   - For AWS
   - For GCP
   - For YC
2. Use the existing NodeGroup resource or create a new one. The `nodeType` parameter for static nodes in the NodeGroup must be set to `Static` or `CloudStatic`.
3. Get the Base64-encoded script code to add and configure the node.

   Here is how you can get Base64-encoded script code to add a node to the `worker` NodeGroup:

   ```
   NODE_GROUP=worker
   d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} -o json | jq
   '.data."bootstrap.sh"' -r
   ```

4. Pre-configure the new node according to the specifics of your environment. For example:
   - Add all the necessary mount points to the `/etc/fstab` file (NFS, Ceph, etc.).
   - Install the necessary packages.
   - Configure network connectivity between the new node and the other nodes of the cluster.
5. Connect to the new node over SSH and run the following command, inserting the Base64 string you got in step 3:

   ```
   echo <Base64-CODE> | base64 -d | bash
   ```

Using the Cluster API Provider Static

A brief example of adding a static node to a cluster using Cluster API Provider Static (CAPS):

1. Prepare the necessary resources.

   - Allocate a server (or a virtual machine), configure networking connectivity, etc. If required, install specific OS packages and add the mount points on the node.
   - Create a user (`caps` in the example below) capable of executing `sudo` by running the following command **on the server**:

     ```
     useradd -m -s /bin/bash caps
     usermod -aG sudo caps
     ```

   - Allow the user to run `sudo` commands without entering a password. For this, add the following line to the `sudo` configuration **on the server** (you can either edit the `/etc/sudoers` file or run the `sudo visudo` command, or use some other method):

```
caps ALL=(ALL) NOPASSWD: ALL
```

- Generate a pair of SSH keys with an empty passphrase **on the server**:

  ```
  ssh-keygen -t ed25519 -f caps-id -C "" -N ""
  ```

  The public and private keys of the `caps` user will be stored in the `caps-id.pub` and `caps-id` files in the current directory on the server.

- Add the generated public key to the `/home/caps/.ssh/authorized_keys` file of the `caps` user by executing the following commands in the keys directory **on the server**:

  ```
  mkdir -p /home/caps/.ssh
  cat caps-id.pub >> /home/caps/.ssh/authorized_keys
  chmod 700 /home/caps/.ssh
  chmod 600 /home/caps/.ssh/authorized_keys
  chown -R caps:caps /home/caps/
  ```

2. Create an SSHCredentials resource in the cluster:

   Run the following command in the user key directory **on the server** to encode the private key to Base64:

   ```
   base64 -w0 caps-id
   ```

   On any computer with `d8 k` configured to manage the cluster, create an environment variable with the value of the Base64-encoded private key you generated in the previous step:

   ```
   CAPS_PRIVATE_KEY_BASE64=<BASE64-ENCODED PRIVATE KEY>
   ```

   To create an SSHCredentials resource in the cluster (note that from this point on, you have to use `d8 k` configured to manage the cluster), run the following command:

   ```
   d8 k create -f - <<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: SSHCredentials
   metadata:
     name: credentials
   spec:
     user: caps
     privateSSHKey: "${CAPS_PRIVATE_KEY_BASE64}"
   EOF
   ```

3. Create a StaticInstance resource in the cluster and specify the IP address of the static node server:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: StaticInstance
   metadata:
     name: static-worker-1
     labels:
       role: worker
   spec:
     # Specify the IP address of the static node server.
     address: "<SERVER-IP>"
     credentialsRef:
       kind: SSHCredentials
       name: credentials
   EOF
   ```

4. Create a NodeGroup resource in the cluster. The value of `count` defines a number of `staticInstances` which fall under the `labelSelector` that will be added to the cluster. In this example it's `1`:

   > **i** The `labelSelector` field in the NodeGroup resource is immutable. To update the `labelSelector`, you need to create a new NodeGroup and move the static nodes into it by changing their labels.

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: worker
EOF
```

Using Cluster API Provider Static for multiple node groups

This example shows how you can use filters in the StaticInstance `label selector` to group static nodes and use them in different NodeGroups. Here, two node groups (`front` and `worker`) are used for different tasks. Each group includes nodes with different characteristics — the `front` group has two servers and the `worker` group has one.

1. Prepare the required resources (3 servers or virtual machines) and create the SSHCredentials resource the same way as on step 1 and step 2 of the example.
2. Create two NodeGroup resources in the cluster (from this point on, use `d8 k` configured to manage the cluster):

   > **i** The `labelSelector` field in the `NodeGroup` resource is immutable. To update the `labelSelector`, you need to create a new NodeGroup and move the static nodes into it by changing their labels.

   ```
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: front
   spec:
     nodeType: Static
     staticInstances:
       count: 2
       labelSelector:
         matchLabels:
           role: front
   ---
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: worker
   spec:
     nodeType: Static
     staticInstances:
       count: 1
       labelSelector:
         matchLabels:
           role: worker
   EOF
   ```

3. Create StaticInstance resources in the cluster and specify the valid IP addresses of the servers:

```
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-front-1
  labels:
    role: front
spec:
  address: "<SERVER-FRONT-IP1>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
---
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-front-2
  labels:
    role: front
spec:
  address: "<SERVER-FRONT-IP2>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
---
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  address: "<SERVER-WORKER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
EOF
```

Adding master nodes in a cloud cluster

To add master nodes in a cloud cluster:

1. Make sure the `control-plane-manager` module is enabled.
2. Open the ClusterConfiguration file (e.g., OpenStackClusterConfiguration).
3. Add or update the `masterNodeGroup` section:

```
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.medium
    imageName: ubuntu-24-04-cloud-amd64
    rootDiskSize: 20
    mainNetwork: default
```

4. Apply the changes using `dhctl converge`:

```
dhctl converge \
  --ssh-host <master node IP> \
  --ssh-user <username> \
  --ssh-agent-private-keys /tmp/.ssh/<key>
```

Using NodeGroup with priority

The `priority` parameter of the NodeGroup custom resource allows you to define the order in which nodes are provisioned in the cluster.
For example, you can configure the cluster to first provision *spot-node* instances and fall back to regular nodes if spot capacity runs out.
Or you can instruct the system to prioritize larger nodes when available, and use smaller ones when resources are limited.

Example of creating two NodeGroups using spot-node instances:

```
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-spot
spec:
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker-spot
    maxPerZone: 5
    minPerZone: 0
    priority: 50
  nodeType: CloudEphemeral
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker
    maxPerZone: 5
    minPerZone: 0
    priority: 30
  nodeType: CloudEphemeral
```

In the example above, the `Cluster Autoscaler` will first attempt to provision a spot-node. If it fails to add the node to the cluster within 15 minutes, the `worker-spot` NodeGroup will be paused for 20 minutes, and the `Cluster Autoscaler` will start provisioning nodes from the `worker` NodeGroup.

If a new node is needed again after 30 minutes, `Cluster Autoscaler` will again attempt `worker-spot` first, then fall back to `worker`.

Once the `worker-spot` NodeGroup reaches its maximum (5 nodes in the example), all further nodes will be provisioned from the `worker` NodeGroup.

The node templates (labels/taints) for `worker` and `worker-spot` NodeGroups should be identical or at least compatible with the workload that triggers scaling.

NodeGroup states and their interpretation

Ready — the NodeGroup contains the minimum required number of scheduled nodes with the status `Ready` for all zones.

Example 1. A group of nodes in the `Ready` state:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 1
status:
  conditions:
  - status: "True"
    type: Ready
---
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    node.deckhouse.io/group: ng1
status:
  conditions:
  - status: "True"
    type: Ready
```

Example 2. A group of nodes in the `Not Ready` state:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 2
status:
  conditions:
  - status: "False"
    type: Ready
---
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    node.deckhouse.io/group: ng1
status:
  conditions:
  - status: "True"
    type: Ready
```

Updating: A node group contains at least one node in which there is an annotation with the prefix `update.node.deckhouse.io` (for example, `update.node.deckhouse.io/waiting-for-approval`).

WaitingForDisruptiveApproval: A node group contains at least one node that has an annotation `update.node.deckhouse.io/disruption-required` and there is no annotation `update.node.deckhouse.io/disruption-approved`.

Scaling: Calculated only for node groups with the type `CloudEphemeral`. The state `True` can be in two cases:

1. When the number of nodes is less than the *target number of nodes* in the group, i.e. when it is necessary to increase the number of nodes in the group.
2. When a node is marked for deletion or the number of nodes is greater than the *target number of nodes*, i.e. when it is necessary to reduce the number of nodes in the group.

The *target number of nodes* is the sum of all replicas in the node group.

Example. The desired number of nodes is 2:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 2
status:
...
  desired: 2
...
```

Error: Contains the last error that occurred when creating a node in a node group.

Additional resources

- Module `cloud-provider-aws` documentation
- Module `cloud-provider-gcp` documentation
- Module `cloud-provider-openstack` documentation
- Module `cloud-provider-yandex` documentation
- Module `control-plane-manager` documentation
- Module `node-manager` documentation

Adding and managing bare-metal nodes

For more details on sizing resources for a bare-metal cluster, see the “Hardware requirements for bare-metal cluster” guide.

Adding nodes to a bare-metal cluster

Manual method

1. Enable the `node-manager` module.
2. Create a NodeGroup object with the type `Static`:

   ```
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: worker
   spec:
     nodeType: Static
   ```

   In this resource, specify the `Static` node type. For all NodeGroup objects in the cluster, Deckhouse automatically generates a `bootstrap.sh` script used to add nodes to the group. When adding nodes manually, you need to copy this script to the server and run it.

   You can obtain the script from the Deckhouse web interface under the “Node Groups → Scripts” tab or via the following `d8 k` command:

   ```
   d8 k -n d8-cloud-instance-manager get secrets manual-bootstrap-for-worker -ojsonpath="{.data.bootstrap\.sh}"
   ```

   The script needs to be decoded from Base64 and then executed as `root`.
3. Once the script finishes, the server will be added to the cluster as a node in the specified group.

Automatic method

⚠ If you have previously increased the number of master nodes in the cluster in the NodeGroup `master` (parameter `spec.staticInstances.count`), before adding nodes using automatic method, make sure that they will not be “captured”.

DKP supports automatic addition of physical (bare-metal) servers to the cluster without the need to manually run an installation script on each node. To enable this:

1. Prepare the server (OS, networking):
   - Install a supported operating system.
   - Configure networking and ensure the server is accessible via SSH.
   - Create a system user (e.g., `ubuntu`) for SSH access.
   - Ensure the user can execute commands using `sudo`.
2. Create an SSHCredentials object to define access to the server. DKP uses this object to connect to the server over SSH. It specifies:
   - A private SSH key encoded in Base64 format.
   - The OS user.
   - The SSH port.
   - (Optional) A `sudo` password, if required.

   Example:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: SSHCredentials
   metadata:
     name: static-nodes
   spec:
     privateSSHKey:
   LS0tLS1CRUdJTiBPUEVOU1NIIFBSSVZBVEUgS0VZLS0tLS0KTUlJRXZBSUJBREFOQmdrcWhraUc5dzBCQVFFRkFBT0N
   BZzhBTUlJQkNnS0NBUUVB
     sshPort: 22
     sudoPassword: password
     user: ubuntu
   ```

   i **Important**. The `privateSSHKey` field must contain a private SSH key encoded in Base64 format. The private key must match the corresponding public key added to the `~/.ssh/authorized_keys` file on the server.
3. Create a StaticInstance` object for each server:

```
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-0
  labels:
    static-node: auto
spec:
  address: 192.168.1.10
  credentialsRef:
    apiVersion: deckhouse.io/v1alpha1
    kind: SSHCredentials
    name: static-nodes
```

A separate StaticInstance resource must be created for each server, but the same SSHCredentials can be reused to access multiple servers.

Possible StaticInstance states:

- `Pending`: The server has not yet been configured; the corresponding node is not present in the cluster.
- `Bootstrapping`: The server is being configured and the node is being added to the cluster.
- `Running`: The server is successfully configured and the node has joined the cluster.
- `Cleaning`: The server is being cleaned up and the node is being removed from the cluster.

These states reflect the current stage of node management. CAPS automatically transitions a StaticInstance between these states depending on whether a node needs to be added or removed from a group.

4. Create a NodeGroup resource describing how DKP should use these servers:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 3
    labelSelector:
      matchLabels:
        static-node: auto
  nodeTemplate:
    labels:
      node-role.deckhouse.io/worker: ""
```

This section defines parameters for using StaticInstance resources:

- `count` specifies how many nodes will be added to the group.
- `labelSelector` defines the rules for selecting nodes.

When using the Cluster API Provider Static (CAPS), it is important to correctly set the `nodeType` to `Static` and provide the `staticInstances` section in the NodeGroup resource:

- If the `labelSelector` is not specified, CAPS will use any available StaticInstance resources in the cluster.
- The same StaticInstance can be used in multiple NodeGroups if it matches the filters.
- CAPS automatically maintains the number of nodes in the group according to the `count` parameter.
- When a node is removed, CAPS performs cleanup and disconnection, and the corresponding StaticInstance transitions to the `Pending` status, allowing it to be reused.

After the node group is created, a script for adding servers to the group will become available. DKP will wait for the required number of StaticInstance objects that match the specified labels. As soon as such an object appears, DKP will use the provided IP address and SSH connection parameters to run the `bootstrap.sh` script and add the server to the group.

Modifying a static cluster configuration

The static cluster settings are stored in the StaticClusterConfiguration structure.

To modify the static cluster parameters, run the following command:

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

Moving a static node between NodeGroups

⚠ During the migration of static nodes between NodeGroups, the node is cleaned up and bootstrapped again, and the `Node` object is recreated.

1. Create a new NodeGroup resource, for example named `front`, which will manage the static node labeled `role: front`:

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: front
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: front
```

2. Change the `role` label of the existing StaticInstance from `worker` to `front`.
   This will allow the new NodeGroup named `front` to manage this node:

```
d8 k label staticinstance static-worker-1 role=front --overwrite
```

3. Update the `worker` NodeGroup resource by decreasing the `count` parameter from `1` to `0`:

```
d8 k patch nodegroup worker -p '{"spec": {"staticInstances": {"count": 0}}}' --type=merge
```

Manual cleanup of a static node

To remove a node from the cluster and clean the server, run the `/var/lib/bashible/cleanup_static_node.sh` script, which is already present on every static node.

Example of disconnecting a node from the cluster and cleaning the server:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
```

> **i** This instruction applies both to nodes manually configured using the bootstrap script and to nodes configured via CAPS.

NodeGroup example

Example NodeGroup definition for static nodes

For virtual machines on hypervisors or physical servers, use static nodes by setting `nodeType: Static` in the NodeGroup.

Example:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
```

Nodes are added to such a group manually using preconfigured scripts or automatically via CAPS.

Changing the CRI for a NodeGroup

CRI (Container Runtime Interface) is a standard interface between the kubelet and the container runtime.

> **⚠** CRI can only be switched between `Containerd` and `NotManaged` via the `cri.type` parameter.

To change the CRI for a NodeGroup, set the `cri.type` parameter to either `Containerd` or `NotManaged`.

Example YAML manifest:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  cri:
    type: Containerd
```

You can also perform this operation using a patch:

- To set `Containerd`:

```
d8 k patch nodegroup <NodeGroup name> --type merge -p '{"spec":{"cri":{"type":"Containerd"}}}'
```

- To set `NotManaged`:

```
d8 k patch nodegroup <NodeGroup name> --type merge -p '{"spec":{"cri":{"type":"NotManaged"}}}'
```

⚠ When changing the `cri.type` for a NodeGroup created using `dhctl`, you must also update this value in `dhctl config edit provider-cluster-configuration` and in the NodeGroup object settings.

After changing the CRI for a NodeGroup, the `node-manager` module will sequentially reboot the nodes, applying the new CRI.
Node updates involve disruption. Depending on the `disruption` settings for the NodeGroup, the `node-manager` module will either automatically update the nodes or require manual approval.

Changing the NodeGroup of a static node

If a node is managed by CAPS, you can't change its associated NodeGroup.
The only option is to delete the StaticInstance and create a new one.

To move an existing manually added static node from one NodeGroup to another, you need to update the group label on the node:

```
d8 k label node --overwrite <node_name> node.deckhouse.io/group=<new_node_group_name>
d8 k label node <node_name> node-role.kubernetes.io/<old_node_group_name>-
```

It will take some time for the changes to take effect.

Changing the IP address in a StaticInstance

You cannot change the IP address of a StaticInstance resource.
If an incorrect address is specified in a StaticInstance, you need to delete the StaticInstance and create a new one.

Deleting a StaticInstance

A StaticInstance in the `Pending` state can be safely deleted without any issues.

To delete a StaticInstance that is in any state other than `Pending` (`Running`, `Cleaning`, `Bootstrapping`), follow these steps:

1. Add the label `"node.deckhouse.io/allow-bootstrap": "false"` to the StaticInstance.

   Example command for adding a label:

   ```
   d8 k label staticinstance d8cluster-worker node.deckhouse.io/allow-bootstrap=false
   ```

2. Wait until the StaticInstance transitions to the `Pending` state.

   To check the status of StaticInstance, use the command:

   ```
   d8 k get staticinstances
   ```

3. Delete the StaticInstance.

   Example command for deleting StaticInstance:

   ```
   d8 k delete staticinstance d8cluster-worker
   ```

4. Decrease the `NodeGroup.spec.staticInstances.count` parameter by 1.

Additional resources

- Global custom resources
- Module `node-manager` documentation

Custom node configuration

To automate actions on group nodes, use the NodeGroupConfiguration resource. It allows you to run bash scripts on the nodes using the Bash Booster command set, as well as apply the Go Template templating engine. This is useful for automating operations such as:

- Installing and configuring additional OS packages.

  Examples:

  - Installing the kubectl plugin
  - Configuring containerd with Nvidia GPU support

- Updating the OS kernel to a specific version.

  Examples:

  - Debian kernel update
  - CentOS kernel update

- Modifying OS parameters.

  Examples:

  - Tuning a sysctl parameter
  - Adding a root certificate

- Collecting information on the node and performing similar tasks.

The NodeGroupConfiguration resource allows you to define execution priority for scripts, limit execution to specific node groups or OS types.

The script code is specified in the `content` field of the resource. When a script is created on a node, the content passes through the Go Template templating engine, which adds a layer of logic to script generation. A dynamic context with a set of variables is available in the template.

Available template variables include:

- `.cloudProvider` (for node groups with `CloudEphemeral` or `CloudPermanent` nodeType) — an array of cloud provider data.

  Example data...

  ```
  cloudProvider:
    instanceClassKind: OpenStackInstanceClass
    machineClassKind: OpenStackMachineClass
    openstack:
      connection:
        authURL: https://cloud.provider.com/v3/
        domainName: Default
        password: p@ssw0rd
        region: region2
        tenantName: mytenantname
        username: mytenantusername
      externalNetworkNames:
      - public
      instances:
        imageName: ubuntu-24-04-cloud-amd64
        mainNetwork: kube
        securityGroups:
        - kube
        sshKeyPairName: kube
      internalNetworkNames:
      - kube
      podNetworkMode: DirectRoutingWithPortSecurityEnabled
    region: region2
    type: openstack
    zones:
    - nova
  ```

- `.cri` — the container runtime interface in use (since Deckhouse version 1.49, only `Containerd` is used).
- `.kubernetesVersion` — the version of Kubernetes in use.
- `.nodeUsers` — an array of user data added to the node using the NodeUser resource.

  Example data...

  ```
  nodeUsers:
  - name: user1
    spec:
      isSudoer: true
      nodeGroups:
      - '*'
      passwordHash: PASSWORD_HASH
      sshPublicKey: SSH_PUBLIC_KEY
      uid: 1050
  ```

- `.nodeGroup` — an array of NodeGroup data.

  Example data...

```
nodeGroup:
  cri:
    type: Containerd
  disruptions:
    approvalMode: Automatic
  kubelet:
    containerLogMaxFiles: 4
    containerLogMaxSize: 50Mi
    resourceReservation:
      mode: "Off"
  kubernetesVersion: "1.31"
  manualRolloutID: ""
  name: master
  nodeTemplate:
    labels:
      node-role.kubernetes.io/control-plane: ""
      node-role.kubernetes.io/master: ""
    taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
  nodeType: CloudPermanent
  updateEpoch: "1699879470"
```

Example of using variables in the template engine:

```
{{- range .nodeUsers }}
echo 'Tuning environment for user {{ .name }}'
# Some code for tuning user environment
{{- end }}
```

Example of using Bash Booster commands:

```
bb-event-on 'bb-package-installed' 'post-install'
post-install() {
  bb-log-info "Setting reboot flag due to kernel was updated"
  bb-flag-set reboot
}
```

Monitoring script execution

Applying NodeGroupConfiguration triggers an update of the **`bashible` configuration of the node group** in the specified groups. DKP automatically detects the change in this configuration and then sets the `UPTODATE` field (in the output of the `d8 k get nodegroup` command) to `0`. The `UPTODATE` field shows how many nodes have already been brought to the **target `bashible` configuration of the node group**; a value of `0` means that no node has been brought to the target `bashible` configuration yet (i.e., the update has not been applied to any node).

DKP controls the start of updates on nodes via the `update.node.deckhouse.io/approved` annotation on the `Node` object. By default, the update runs simultaneously on one node from each group. Update parallelism is defined by the `maxConcurrent` parameter in the node group configuration. When DKP selects a node to update (taking the queue and `maxConcurrent` into account), it sets the `update.node.deckhouse.io/approved` annotation, after which the `bashible` service on that node begins applying the **target `bashible` configuration of the node group**.

You can view the node annotations set by DKP to start the update using the following command:

```
d8 k get nodes -o json | jq '.items[] |
select(.metadata.annotations."update.node.deckhouse.io/approved"=="") | .metadata.name' -r
```

You can monitor script execution on the node in the `bashible` service logs using:

```
journalctl -fu bashible.service
```

The scripts are located in the `/var/lib/bashible/bundle_steps/` directory on the node.

Script re-execution mechanism

The service decides whether to re-run the scripts by comparing a unified checksum of all files, located at `/var/lib/bashible/configuration_checksum`, with the checksum stored in the `configuration-checksums` secret in the `d8-cloud-instance-manager` namespace in the Kubernetes cluster.

You can check the checksum with the following command:

```
d8 k -n d8-cloud-instance-manager get secret configuration-checksums -o yaml
```

The checksum comparison is performed by the service every minute.

The checksum in the cluster is updated every 4 hours, thereby re-triggering the execution of scripts on all nodes.

To manually trigger `bashible` execution on a node, you can delete the checksum file using the following command:

```
rm /var/lib/bashible/configuration_checksum
```

Script writing specifics

When writing scripts, it’s important to consider the following features of their usage in Deckhouse:

1. Scripts in DKP are executed every 4 hours or based on external triggers. Therefore, it’s important to write scripts in a way that they first check whether changes are necessary, to avoid repeated or unnecessary actions on each execution.
2. There are predefined scripts that perform various actions, including service installation and configuration. It’s important to consider this when assigning priority to custom scripts. For example, if a custom script restarts a service, it must run after the script that installs that service. Otherwise, the custom script won’t be able to run during the initial provisioning of the node (since the service won’t be installed yet).

Useful specifics of certain scripts:

- `032_configure_containerd.sh`: Merges the containerd configuration files and **restarts** the service. The location of the configuration files depends on the CRI used on the cluster nodes:
  - `/etc/containerd/conf.d/*.toml`: For containerd v1.
  - `/etc/containerd/conf2.d/*.toml`: For containerd v2.

  Note that directories containing configuration files are not created automatically. Files must be added to these directories in scripts with a priority of less than `32`.

How to use containerd with Nvidia GPU support

You need to create a separate NodeGroup for GPU nodes:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: gpu
spec:
  chaos:
    mode: Disabled
  disruptions:
    approvalMode: Automatic
  nodeType: CloudStatic
```

Next, create a NodeGroupConfiguration resource for the `gpu` NodeGroup to configure `containerd`:

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

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/nvidia_gpu.toml - << "EOF"
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".containerd]
          default_runtime_name = "nvidia"
          [plugins."io.containerd.grpc.v1.cri".containerd.runtimes]
            [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
              [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.nvidia]
                privileged_without_host_devices = false
                runtime_engine = ""
                runtime_root = ""
                runtime_type = "io.containerd.runc.v2"
                [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.nvidia.options]
                  BinaryName = "/usr/bin/nvidia-container-runtime"
                  SystemdCgroup = false
    EOF
  nodeGroups:
  - gpu
  weight: 31
```

Add a NodeGroupConfiguration resource to install Nvidia drivers for the `gpu` NodeGroup.

Ubuntu

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

    if [ ! -f "/etc/apt/sources.list.d/nvidia-container-toolkit.list" ]; then
      distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
      curl -s -L https://nvidia.github.io/libnvidia-container/gpgkey | sudo apt-key add -
      curl -s -L https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-container.list
| sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
    fi
    bb-apt-install nvidia-container-toolkit nvidia-driver-535-server
    nvidia-ctk config --set nvidia-container-runtime.log-level=error --in-place
  nodeGroups:
  - gpu
  weight: 30
```

CentOS

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

    if [ ! -f "/etc/yum.repos.d/nvidia-container-toolkit.repo" ]; then
      distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
      curl -s -L https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-container.repo
| sudo tee /etc/yum.repos.d/nvidia-container-toolkit.repo
    fi
    bb-dnf-install nvidia-container-toolkit nvidia-driver
    nvidia-ctk config --set nvidia-container-runtime.log-level=error --in-place
  nodeGroups:
  - gpu
  weight: 30
```

After the configurations are applied, bootstrap and reboot the nodes to apply the settings and install the drivers.

MIG in a single-node cluster

> **i** General information about preparing GPU nodes, installing drivers, and verifying GPU operation is provided in the How do I work with GPU nodes section.

In a single-node cluster that uses the `MIG` strategy, the MIG configuration must be reapplied manually after each node reboot.

This limitation applies to the current MIG implementation, which uses the `nvidia-mig-manager` component.

After each node reboot, perform the following steps:

1. Add the service annotation to the node:

   ```
   d8 k annotate node <node_name> update.node.deckhouse.io/drained=""
   ```

2. Restart the `nvidia-mig-manager` component:

   ```
   d8 k -n d8-nvidia-gpu delete pod -l component=nvidia-mig-manager
   ```

3. Wait until the MIG configuration is reapplied by checking the `nvidia-mig-manager` logs:

   ```
   d8 k -n d8-nvidia-gpu logs -f -l component=nvidia-mig-manager
   ```

   Successful configuration application is confirmed by a log entry similar to:

   ```
   Successfully updated to MIG config: <partedConfig>
   ```

   where `<partedConfig>` is the value of the `partedConfig` parameter in the NodeGroup resource (for example, `all-1g.10gb`).

4. After the MIG configuration has been successfully applied, return the node to the schedulable state:

   ```
   d8 k uncordon <node_name>
   ```

Verifying successful installation

Create the following Job in your cluster:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: nvidia-cuda-test
  namespace: default
spec:
  completions: 1
  template:
    spec:
      restartPolicy: Never
      nodeSelector:
        node.deckhouse.io/group: gpu
      containers:
        - name: nvidia-cuda-test
          image: nvidia/cuda:11.6.2-base-ubuntu20.04
          imagePullPolicy: "IfNotPresent"
          command:
            - nvidia-smi
```

Check the logs with the following command:

```
d8 k logs job/nvidia-cuda-test
```

Example output:

```
Tue Jan 24 11:36:18 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.60.13    Driver Version: 525.60.13    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:8B:00.0 Off |                    0 |
| N/A   45C    P0    25W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Create the following Job in your cluster:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: gpu-operator-test
  namespace: default
spec:
  completions: 1
  template:
    spec:
      restartPolicy: Never
      nodeSelector:
        node.deckhouse.io/group: gpu
      containers:
        - name: gpu-operator-test
          image: nvidia/samples:vectoradd-cuda10.2
          imagePullPolicy: "IfNotPresent"
```

Check the logs with the following command:

```
d8 k logs job/gpu-operator-test
```

Example output:

```
[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
```

How to deploy a custom containerd configuration file

✖ Adding custom settings will trigger a restart of the `containerd` service.

Bashible on nodes merges main Deckhouse containerd config with configs from:

- `/etc/containerd/conf.d/*.toml` — if containerd v1 is used as the CRI on the cluster nodes.
- `/etc/containerd/conf2.d/*.toml` — if containerd v2 is used as the CRI on the cluster nodes.

⚠ You can override the parameter values defined in the `/etc/containerd/deckhouse.toml` file. However, you are responsible for ensuring the correct operation of such changes. It is recommended **not to modify the configuration** on control plane (master) nodes (NodeGroup `master`).

- For containerd v1
- For containerd v2

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-option-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2024 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_option.toml - << EOF
    oom_score = 500
    [metrics]
    address = "127.0.0.1"
    grpc_histogram = true
    EOF
  nodeGroups:
    - "worker"
  weight: 31
```

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-option-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2024 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf2.d
    bb-sync-file /etc/containerd/conf2.d/runtimeclass.toml - << "EOF"
    [plugins."io.containerd.cri.v1.runtime".containerd.runtimes.gpu-large-shm]
      runtime_type = "io.containerd.runc.v2"
      [plugins."io.containerd.cri.v1.runtime".containerd.runtimes.gpu-large-shm.options]
        BinaryName = ""
        SystemdCgroup = true
        ShmSize = 17179869184
    EOF
  nodeGroups:
    - "worker"
  weight: 31
```

Adding a configuration for an additional registry

Containerd supports two methods for registry configuration: the **deprecated** method and the **actual** method.

To check for the presence of the **deprecated** configuration method, run the following commands on the cluster nodes:

```
cat /etc/containerd/config.toml | grep 'plugins."io.containerd.grpc.v1.cri".registry.mirrors'
cat /etc/containerd/config.toml | grep 'plugins."io.containerd.grpc.v1.cri".registry.configs'
```

Example output:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors]
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."<REGISTRY_URL>"]
[plugins."io.containerd.grpc.v1.cri".registry.configs]
  [plugins."io.containerd.grpc.v1.cri".registry.configs."<REGISTRY_URL>".auth]
```

To check for the presence of the **actual** configuration method, run the following command on the cluster nodes:

```
cat /etc/containerd/config.toml | grep '/etc/containerd/registry.d'

# Example output:
# config_path = "/etc/containerd/registry.d"
```

Old Method

⚠ This containerd configuration format is deprecated.

i Used in containerd v1 when Deckhouse is not managed by the Registry module.

The configuration is described in the main containerd configuration file `/etc/containerd/config.toml`.

Adding custom configuration is carried out through the `toml merge` mechanism. Configuration files from the `/etc/containerd/conf.d` directory are merged with the main file `/etc/containerd/config.toml`. The merge takes place during the execution of the `032_configure_containerd.sh` script, so the corresponding files must be added in advance.

Example configuration file for the `/etc/containerd/conf.d/` directory:

```
[plugins]
  [plugins."io.containerd.grpc.v1.cri"]
    [plugins."io.containerd.grpc.v1.cri".registry]
      [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
          endpoint = ["https://${REGISTRY_URL}"]
      [plugins."io.containerd.grpc.v1.cri".registry.configs]
        [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
          auth = "${BASE_64_AUTH}"
          username = "${USERNAME}"
          password = "${PASSWORD}"
        [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
          ca_file = "${CERT_DIR}/${CERT_NAME}.crt"
          insecure_skip_verify = true
```

✖ Adding custom settings through the `toml merge` mechanism causes the containerd service to restart.

Adding authorization to an additional registry (deprecated method)

Example of adding authorization to an additional registry when using the **deprecated** configuration method:

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

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
              endpoint = ["https://${REGISTRY_URL}"]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]
            [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
              username = "username"
              password = "password"
              # OR
              auth = "dXNlcm5hbWU6cGFzc3dvcmQ="
    EOF
```

Configuring a certificate for an additional registry (deprecated method)

Example of configuring a certificate for an additional registry when using the **deprecated** configuration method:

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

    REGISTRY_URL=private.registry.example
    CERT_FILE_NAME=${REGISTRY_URL}
    CERTS_FOLDER="/var/lib/containerd/certs/"


    mkdir -p ${CERTS_FOLDER}
    bb-sync-file "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" - << EOF
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    EOF

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]
            [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
              ca_file = "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"
    EOF
```

> **i** In addition to containerd, the certificate can be added to the OS.

Adding TLS skip verify (deprecated method)

Example of adding TLS skip verify when using the **deprecated** configuration method:

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

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]
            [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
              insecure_skip_verify = true
    EOF
```

After applying the configuration file, verify access to the registry from the nodes using the command:

```
# Via the CRI interface
crictl pull private.registry.example/image/repo:tag
```

Configuring a mirror for access to public registries (deprecated method)

Example of configuring a mirror for public image registries when using the **deprecated** configuration method:

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

    sed -i '/endpoint = \["https:\/\/registry-1.docker.io"\]/d'
/var/lib/bashible/bundle_steps/032_configure_containerd.sh
    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/mirror-to-harbor.toml - << "EOF"
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
              endpoint = ["https://registry.private.network/v2/dockerhub-proxy/"]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."gcr.io"]
              endpoint = ["https://registry.private.network/v2/YOUR_GCR_PROXY_REPO/"]
    EOF
```

New Method

> **i**
>
> Used in containerd v2.
>
> Used in containerd v1 when managed through the `registry` module (for example, in `Direct` mode).

The configuration is defined in the `/etc/containerd/registry.d` directory.
Configuration is specified by creating subdirectories named after the registry address:

```
/etc/containerd/registry.d
├── private.registry.example:5001
│   ├── ca.crt
│   └── hosts.toml
└── registry.deckhouse.io
    ├── ca.crt
    └── hosts.toml
```

Example contents of the `hosts.toml` file:

```
[host]
  # Mirror 1.
  [host."https://${REGISTRY_URL_1}"]
    capabilities = ["pull", "resolve"]
    ca = ["${CERT_DIR}/${CERT_NAME}.crt"]

    [host."https://${REGISTRY_URL_1}".auth]
      username = "${USERNAME}"
      password = "${PASSWORD}"

  # Mirror 2.
  [host."http://${REGISTRY_URL_2}"]
    capabilities = ["pull", "resolve"]
    skip_verify = true
```

> **i** Configuration changes do not cause the containerd service to restart.

Adding authorization to an additional registry (actual method)

Example of adding authorization to an additional registry when using the **actual** configuration method:

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

    REGISTRY_URL=private.registry.example

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

Configuring a certificate for an additional registry (actual method)

Example of configuring a certificate for an additional registry when using the **actual** configuration method:

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

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"

    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/ca.crt" - << EOF
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    EOF

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

i In addition to containerd, the certificate can be added to the OS.

Adding TLS skip verify (actual method)

Example of adding TLS skip verify when using the **actual** configuration method:

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

    REGISTRY_URL=private.registry.example

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

After applying the configuration file, check access to the registry from the nodes using the following commands:

```
# Via the CRI interface.
crictl pull private.registry.example/image/repo:tag

# Via ctr with the configuration directory specified.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/ private.registry.example/image/repo:tag

# Via ctr for an HTTP registry.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/ --plain-http private.registry.example/image/repo:tag
```

Configuring a mirror for access to public registries (actual method)

Example of configuring a mirror for public image registries when using the **actual** configuration method:

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

    REGISTRY1_URL=docker.io
    mkdir -p "/etc/containerd/registry.d/${REGISTRY1_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY1_URL}/hosts.toml" - << EOF
    [host."https://registry.private.network/v2/dockerhub-proxy/"]
      capabilities = ["pull", "resolve"]
      override_path = true
    EOF
    REGISTRY2_URL=gcr.io
    mkdir -p "/etc/containerd/registry.d/${REGISTRY2_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY2_URL}/hosts.toml" - << EOF
    [host."https://registry.private.network/v2/dockerhub-proxy/"]
      capabilities = ["pull", "resolve"]
      override_path = true
    EOF
```

How to automatically put custom labels on the node

1. On the node, create the directory `/var/lib/node_labels`.
2. In this directory, create a single of multiple files containing the necessary labels. There can be any number of files, as well as the number of subdirectories containing them.
3. Add the necessary labels to the files in the `key=value` format. For example:

   ```
   example-label=test
   ```

4. Save the files.

When adding a node to the cluster, the labels specified in the files will be automatically affixed to the node.

> ⚠ Note that it is not possible to add labels used in DKP in this way. This method will only work with custom labels that do not overlap with those reserved for Deckhouse.

Additional resources

- Module `deckhouse` parameters
- Module `node-manager` documentation
- Module `registry` documentation

Migrating container runtime to containerd v2

You can configure containerd v2 as the primary container runtime either at the cluster level or for specific node groups. This runtime option enables the use of cgroups v2, provides improved security, and allows more flexible resource management.

Requirements

Migration to containerd v2 is possible under the following conditions:

- Nodes meet the requirements described in the cluster-wide parameters.
- The node kernel version is not in the range 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- There are no custom configurations on the server in `/etc/containerd/conf.d` (example of a custom configuration).

> ⚠ On Ubuntu nodes in AWS, the `erofs` kernel module required by containerd v2 is shipped in a separate `linux-modules-extra` package. DKP installs it automatically, but for some kernels, mostly outdated ones, the package is no longer published in the APT repository. In that case, the installation is skipped, a warning is written to the node logs, the node gets the `node.deckhouse.io/containerd-v2-unsupported` label, and the `D8NodeContainerdV2NotSupported` alert fires.
>
> The problem has been confirmed for the `5.15.0-1028-aws`, `6.8.0-1024-aws`, and `6.8.0-1029-aws` kernels. To migrate such nodes, switch the node group to a newer AMI or kernel that provides `erofs`.
>
> Before migrating, check whether there are such nodes in the cluster: check the alerts and the list of labelled nodes (see the commands below).

If any of the requirements described in the general cluster parameters are not met, Deckhouse Kubernetes Platform adds the label `node.deckhouse.io/containerd-v2-unsupported` to the node. If the node has custom configurations in the `/etc/containerd/conf.d/` (if CRI containerd v1 is used on the cluster nodes) or `/etc/containerd/conf2.d/` (if CRI containerd v2 is used on the cluster nodes) directories, the label `node.deckhouse.io/containerd-config=custom` is added to it.

If one of these labels is present, changing the `spec.cri.type` parameter for the node group will be unavailable. Nodes that do not meet the migration conditions can be viewed using the following commands:

```
d8 k get node -l node.deckhouse.io/containerd-v2-unsupported
d8 k get node -l node.deckhouse.io/containerd-config=custom
```

Additionally, a administrator can verify if a specific node meets the requirements using the following commands:

```
uname -r | cut -d- -f1
stat -f -c %T /sys/fs/cgroup
systemctl --version | awk 'NR==1{print $2}'
modprobe -qn erofs && echo "TRUE" || echo "FALSE"
ls -l /etc/containerd/conf.d
```

How to enable containerd v2

> ℹ During migration the directory `/var/lib/containerd` will be cleared, causing all pod images to be re-pulled, and the node will reboot.
>
> Also note that this is a disruptive update. If `spec.disruptions.approvalMode: Manual` is set for a node group, approve the update on the respective nodes with the annotation: `d8 k annotate node <NODE-NAME> update.node.deckhouse.io/disruption-approved=`.

You can enable containerd v2 in two ways:

1. **For the entire cluster**. Set the value `ContainerdV2` for the `defaultCRI` parameter in the `ClusterConfiguration` resource. This value will apply to all NodeGroup objects where `spec.cri.type` is not explicitly defined.

   Example:

   ```
   apiVersion: deckhouse.io/v1
   kind: ClusterConfiguration
   ...
   defaultCRI: ContainerdV2
   ```

2. **For a specific node group**. Set `ContainerdV2` in the `spec.cri.type` parameter of the NodeGroup object.

   Example:

   ```
   apiVersion: deckhouse.io/v1
   kind: NodeGroup
   metadata:
     name: worker
   spec:
     cri:
       type: ContainerdV2
   ```

   When migrating to containerd v2 Deckhouse Kubernetes Platform will begin sequentially updating the nodes.

Additional resources

- Global custom resources
- Module `node-manager` documentation

Application scaling and resource management

Scaling applications and managing pod placement in Deckhouse Kubernetes Platform allows dynamic adaptation of cluster resources to the current load, ensuring high availability of services and efficient resource utilization.

Deckhouse supports all major Kubernetes scaling and workload distribution mechanisms:

- Horizontal Pod Autoscaling (HPA) — automatic adjustment of the number of pod replicas based on resource consumption metrics or external indicators.
- Vertical Pod Autoscaling (VPA) — automatic tuning of requested CPU and memory resources for containers based on actual usage.
- Scaling by metrics — using arbitrary metrics to flexibly manage application scaling through Prometheus.
- Pod redistribution (Descheduler) — automatically evicting pods to optimize workload placement across the cluster.
- Pod priorities (Priority Classes) — managing the eviction order of pods during resource shortages based on their importance.
- Scheduler — configuring the rules and logic for node selection when placing pods.

Deckhouse Kubernetes Platform enables automatic application scaling and efficient resource management to ensure stable and predictable cluster operation.

The following sections provide a detailed description of scaling capabilities, configuration examples, and recommendations for their effective use.

Scaling by metrics

Scaling based on metrics

Metric-based scaling is the process of automatically or manually adjusting resources (e.g., number of pod replicas, allocated CPU/memory) based on defined metrics. These metrics can range from CPU or memory usage to the number of requests per second or the size of a message queue.

Scaling can be based on both standard and custom monitoring metrics. For example, scaling can be triggered by the following metrics:

- Pod CPU — current CPU usage.
- Pod Memory — current memory usage.
- RPS (Ingress) — number of requests per second over 1, 5, or 15 minutes (`rps 1m`, `rps_5m`, `rps_15m`).
- Average CPU usage (Pod) — over 1, 5, or 15 minutes (`cpu 1m`, `cpu 5m`, `cpu 15m`).
- Average memory usage (Pod) — over 1, 5, or 15 minutes (`memory_1m`, `memory_5m`, `memory_15m`).

Configuration

To simplify metric-based scaling configuration, DKP offers special resources (`Cluster*Metric` and `*Metric`). You can also configure scaling based on custom metric calculation rules.

To configure scaling based on metrics, follow these steps:

1. Enable the `prometheus-metrics-adapter` module. It allows you to define custom resources that describe metric calculation rules using PromQL queries.

   You can enable the `prometheus-metrics-adapter` module in the Deckhouse web interface or via the following command:

   ```
   d8 system module enable prometheus-metrics-adapter
   ```

2. Define a PromQL query that describes the desired metric (e.g., application request rate or CPU usage over time). This query is registered in the cluster as a metric for HPA.
3. Create a HorizontalPodAutoscaler) resource, specifying the metric name and threshold value. Kubernetes will query the result of your PromQL expression and compare it to the threshold, adjusting the number of replicas as needed (see the HPA configuration documentation for more details).

Example PromQL query:

```
query: sum(rate(ingress_nginx_detail_requests_total{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>) OR
on() vector(0)
```

In this example:

- `ingress_nginx_detail_requests_total` — the base metric (a request counter, e.g., from the Ingress NGINX controller);
- `rate(...[2m])` — calculates the rate of increase over the past 2 minutes (requests per second);
- `sum(...) by (<<.GroupBy>>)` — aggregates the results by a set of labels (e.g., Ingress name, namespace, etc.);
- `OR on() vector(0)` — a PromQL construct that ensures the result is `0` when there’s no data (otherwise, the query might return nothing, which would be ignored).

This PromQL query effectively provides the current requests per second value filtered by specified labels (Ingress, namespace, etc.).

Once an administrator defines a metric using a PromQL query (e.g., in an IngressMetric object or a similar mechanism), it can be referenced in a HorizontalPodAutoscaler (HPA) as shown below:

```
apiVersion: deckhouse.io/v1beta1
kind: IngressMetric
metadata:
  name: mymetric
  namespace: mynamespace
spec:
  query: sum(rate(ingress_nginx_detail_requests_total{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>) OR
on() vector(0)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # The controller to scale (reference to a Deployment or StatefulSet).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 1
  maxReplicas: 2
  # Metrics used for scaling.
  metrics:
  - type: Object
    object:
      # The object that owns the metrics in Prometheus.
      describedObject:
        apiVersion: networking.k8s.io/v1
        kind: Ingress
        name: myingress
      metric:
        # The metric registered via a custom resource like IngressMetric or ClusterIngressMetric.
        # You can also use built-in metrics like rps_1m, rps_5m, or rps_15m provided by the
prometheus-metrics-adapter module.
        name: mymetric
      target:
        # For Object-type metrics, use either `Value` or `AverageValue`.
        type: AverageValue
        # Scaling will occur if the average value of the metric across all pods in the Deployment
differs significantly from 10.
        averageValue: 10
```

i When the metric exceeds (or falls below) the threshold `averageValue: 10`, the system will adjust the number of replicas of `myapp` within the range of 1 to 2.

Configuring metrics using observability module resources

To configure custom metric calculation rules:

1. Make sure the `observability` module is enabled.
2. Create a ClusterObservabilityMetricsRulesGroup or ObservabilityMetricsRulesGroup resource:
   - ClusterObservabilityMetricsRulesGroup: Intended for cluster-wide rules applied to system namespaces.
   - ObservabilityMetricsRulesGroup: Intended for rules that operate within a specific namespace.

Configuration example:

```
apiVersion: observability.deckhouse.io/v1alpha1
kind: ClusterObservabilityMetricsRulesGroup
metadata:
  name: custom-cluster-metrics-mymetric
spec:
  interval: 1m
  rules:
    - record: kube_adapter_metric_ingress_bytes_sent
      expr: |
        sum(ingress_nginx_detail_sent_bytes_sum) by (namespace, ingress)
```

Working with unstable metrics

When dealing with unstable metrics (e.g., metrics that fluctuate and cause excessive scaling), it is recommended to:

- Use PromQL aggregation functions. For example, `avg_over_time()` smooths the metric over a specified time range, helping to avoid sudden spikes:

```
apiVersion: deckhouse.io/v1beta1
kind: ServiceMetric
metadata:
  name: rmq-queue-forum-messages
  namespace: mynamespace
spec:
  query: sum
(avg_over_time(rabbitmq_queue_messages{<<.LabelMatchers>>,queue=~"send_forum_message",vhost="/"
}[5m])) by (<<.GroupBy>>)
```

- Configure stabilization behavior in the autoscaler. You can increase the stabilization window so that decisions to scale up or down are based on more stable data.

Retrieving metric values

To get a list of available metrics, use the following command:

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/
```

To retrieve metric values associated with specific objects, use:

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-namespace/services/*/my-service-
metric
```

To get values of metrics created via `NamespaceMetric`, use:

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-namespace/metrics/my-ns-metric
```

To retrieve external metrics, use:

```
d8 k get --raw /apis/external.metrics.k8s.io/v1beta1
```

Configuring autoscaling in the Deckhouse Web Interface

In Deckhouse Kubernetes Platform, you can configure node autoscaling settings through the Deckhouse web interface. This allows you to dynamically change the number of nodes depending on the load.

To configure autoscaling in the web interface:

- Open the Deckhouse web interface;
- In the left menu, go to the “Nodes” → “Node Groups” section;
- Select the desired NodeGroup;
- In the “Autoscaling Parameters” section, the current settings will be displayed.

The following settings are available in the Deckhouse web interface:

- Nodes per zone – sets the minimum and maximum number of nodes that can run in a single zone.
- Desired – the current number of nodes needed for operation.
- Requested – the number of nodes currently scheduled for provisioning.
- Standby – additional reserved nodes.

To modify autoscaling parameters:

- Click the edit icon next to the “Nodes per zone” parameter.
- In the pop-up window, specify the minimum and maximum number of nodes.
- Click “Apply” to save the changes, or “Cancel” to discard them.

When you click “Create based on” in the web interface, a form for configuring autoscaling parameters opens:

1. **Main parameters:**
   - Zones – selection of availability zones where machines will be created. This helps distribute nodes across multiple zones. The default value depends on the selected cloud provider and usually includes all zones in the region.
   - Scaling priority – used if multiple node groups exist with the same priority. If not specified, the system will randomly choose a group for scaling.
   - Number of machines created simultaneously during scale-up – defines how many machines can be added at once during scale-up. For example, if the value is 1, nodes will be added one at a time.
   - Max unavailable instances during RollingUpdate – how many nodes can be unavailable during updates. A value of 0 means rolling updates will occur one by one with no downtime.
2. **Standby nodes and resources:**
   - Number of standby nodes out of total nodes – you can specify an exact number (e.g., 2–6) or a percentage (e.g., 15%). A standby node is a pre-provisioned node ready to receive workloads immediately. This speeds up scale-up by several minutes.
   - Resources reserved on standby nodes – the percentage of resources to reserve on standby nodes. Acceptable values: 1% to 80%. For example, if set to 50%, half of the node’s resources will be reserved.
3. **Additional parameters:**
   - Quick Shutdown – enables fast node shutdown, reducing the drain wait time to 5 minutes.

Additional resources

- Module `console` documentation
- Module `observability` documentation
- Module `prometheus-metrics-adapter` documentation

Horizontal pod autoscaling

How Horizontal Scaling (HPA) works

Horizontal Pod Autoscaler (HPA) is a mechanism for automatically adjusting (up or down) the number of pod replicas (in Deployments or StatefulSets) based on metrics retrieved via the Kubernetes API. HPA monitors application load by checking current metrics (e.g., CPU, memory, or custom Prometheus metrics), and adjusts the number of replicas as needed to maintain a desired level of performance or to optimize resource usage.

Available metric types for HPA

Horizontal scaling in DKP can be based on any available metrics, such as:

1. Pod CPU and memory usage.
   - Configured using the HorizontalPodAutoscaler resource.
     For example, you can define a Resource-type metric with `averageUtilization = 70` for CPU, so that the application scales up when average CPU usage reaches 70%.
2. DKP object metric (Ingress, Service) or pod-based metrics (sum or average across all pods of a controller).
   - Enables scaling based on metrics attached to DKP objects (e.g., Ingress, Service), or metrics aggregated from pods (sum or average per controller). Resources like ServiceMetric and IngressMetric are used for this.
3. Any other metrics, including external data (such as Amazon SQS metrics, cloud load balancers, SaaS services, etc.).
   - Useful when the metric source is outside the cluster. Metric calculation rules are configured using the ClusterObservabilityMetricsRulesGroup resource (for cluster-wide metrics) and the ObservabilityMetricsRulesGroup resource (for namespaced metrics) and require enabling the `observability` module.

HPA recommendations

If metrics fluctuate, wrap the metric in an aggregation function (e.g., `avg_over_time()`) or increase the stabilization window (`spec.behavior.scaleUp.stabilizationWindowSeconds`) to avoid rapid scaling of pods.

HPA limitations

1. By default, HPA handles scaling up and scaling down differently:
   - Scaling up:
     - Happens immediately (`spec.behavior.scaleUp.stabilizationWindowSeconds = 0`).
     - Limit — within 15 seconds, the number of pods can double. If there were fewer than 4 pods, up to 4 new pods can be added.
   - Scaling down:
     - Takes up to 5 minutes (`spec.behavior.scaleDown.stabilizationWindowSeconds = 300`).
     - Multiple “proposals” for a new number of replicas are collected, and the highest one is selected to avoid frequent downsizing.
     - There are no limits on how many pods can be removed at once.
2. Only one HPA per controller. You cannot assign multiple HPAs to the same Deployment (or StatefulSet) — they will conflict.

How to enable or disable HPA

HPA does not require separate activation in DKP. However, if you want to scale based on metrics other than CPU and memory, you must enable the `prometheus-metrics-adapter` module. See how to enable it in the documentation.

HPA configuration

To configure HPA, follow these steps:

1. Define the controller (Deployment or StatefulSet) to scale.
2. Set scaling limits (`minReplicas` and `maxReplicas`):

   ```
   minReplicas: 1
   maxReplicas: 10
   ```

3. Configure metrics:

   ```
   metrics:
   - type: Resource
     resource:
       name: cpu
       target:
         type: Utilization
         averageUtilization: 70
   ```

4. Optionally, set the `stabilizationWindowSeconds` to delay scaling decisions and limit the pod growth rate:

Example for scaling up:

```
behavior:
 scaleUp:
   stabilizationWindowSeconds: 300
```

Example for scaling down:

```
behavior:
 scaleDown:
   stabilizationWindowSeconds: 300
```

HPA configuration examples

Scaling based on CPU and memory

Scaling occurs when the average CPU or memory usage across all pods exceeds the specified percentage:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: app-hpa
  namespace: app-prod
spec:
  # Specifies the controller to scale (reference to a Deployment or StatefulSet).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: app
  # Scaling boundaries for the controller.
  minReplicas: 1
  maxReplicas: 10
  # If the application tends to have short CPU usage spikes,
  # you can delay the scaling decision to confirm it's necessary.
  # By default, scaling up happens immediately.
  behavior:
    scaleUp:
      stabilizationWindowSeconds: 300
  metrics:
  # CPU and memory-based scaling.
  - type: Resource
    resource:
      name: cpu
      target:
        # Scaling happens when the average CPU utilization across all pods in scaleTargetRef exceeds
this value.
        # For metrics of type: Resource, only target type: Utilization is available.
        type: Utilization
        # Example: if each pod requests 1 core, scaling occurs when average usage exceeds 700m.
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        # Scaling based on memory usage exceeding a certain percentage.
        type: Utilization
        # Example: if each pod requests 1 GiB, scaling occurs when average usage exceeds 800 MiB.
        averageUtilization: 80
```

Scaling based on object metrics

If the `rmq-queue-forum-messages` metric (number of messages in RabbitMQ) exceeds 42, the HPA increases the number of replicas in the `myconsumer` Deployment:

```
apiVersion: deckhouse.io/v1beta1
kind: ServiceMetric
metadata:
  name: rmq-queue-forum-messages
  namespace: mynamespace
spec:
  query: sum (rabbitmq_queue_messages{<<.LabelMatchers>>,queue=~"send_forum_message",vhost="/"}) by
(<<.GroupBy>>)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Specifies the controller to scale (reference to a Deployment or StatefulSet).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myconsumer
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Object
    object:
      describedObject:
        apiVersion: v1
        kind: Service
        name: rmq
      metric:
        name: rmq-queue-forum-messages
      target:
        type: Value
        value: 42
```

Scaling based on external metrics

The `mymetric` metric is sourced from an external system (e.g., SQS). Scaling is triggered when the metric value exceeds 100:

```
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Specifies the controller to scale (reference to a Deployment or StatefulSet).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 1
  maxReplicas: 2
  metrics:
  # Using external metrics for scaling.
  - type: External
    external:
      metric:
        # This is the metric registered in Prometheus as kube_adapter_metric_mymetric,
        # but used here without the 'kube_adapter_metric_' prefix.
        name: mymetric
        selector:
          # Use label selectors to narrow down the metric query.
          matchLabels:
            namespace: mynamespace
            ingress: myingress
      target:
        # Only `type: Value` is supported for External metrics.
        type: Value
        # Scaling is triggered if the metric value exceeds 10.
        value: 10
```

Additional resources

- Module `observability` documentation
- Module `prometheus-metrics-adapter` documentation

Vertical pod autoscaling

How Vertical Scaling (VPA) works

Vertical Pod Autoscaler (VPA) automates container resource management and significantly improves application efficiency. VPA is useful in scenarios where the exact resource requirements of an application are unknown in advance. When VPA is enabled and the appropriate operating mode is set, the requested resources are determined based on actual usage metrics gathered from the monitoring system. It is also possible to configure the system to only recommend resource settings without applying them automatically.

If the application load changes depending on time of day, user requests, or other factors, VPA automatically adjusts the allocated resources. This helps prevent outages due to lack of resources or excessive CPU and memory consumption.

For details on VPA operating modes, its components and limitations, refer to Architecture.

How to enable or disable VPA

You can enable VPA in the following ways:

1. Via a ModuleConfig resource (e.g., ModuleConfig/vertical-pod-autoscaler). Set the `spec.enabled` parameter to `true` or `false`:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: vertical-pod-autoscaler
   spec:
     enabled: true
   ```

2. Via the `d8` command (run inside the `d8-system/deckhouse pod`):

   ```
   d8 system module enable vertical-pod-autoscaler
   ```

3. Through the Deckhouse web interface:

   - Go to the “Deckhouse → Modules” section;
   - Find the vertical-pod-autoscaler module and click on it;
   - Toggle the “Module enabled” switch.

The `vertical-pod-autoscaler` module has no required configuration — it can be enabled and used with default settings.

After creating a VerticalPodAutoscaler resource, you can check VPA recommendations using the following command:

```
d8 k describe vpa my-app-vpa
```

In the `status` section, you’ll see the following parameters:

- `Target`: The recommended resource amount for the pod.
- `Lower Bound`: The minimum recommended resource amount for the application.
- `Upper Bound`: The maximum recommended resource amount for the application.
- `Uncapped Target`: The resource value based on extreme metrics without considering historical data.

VPA configuration

1. Create the VPA module configuration.

   To configure VPA, you need to create a configuration file for the module. Example configuration:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: vertical-pod-autoscaler
   spec:
     version: 1
     enabled: true
     settings:
       nodeSelector:
         node-role/system: ""
       tolerations:
       - key: dedicated.deckhouse.io
         operator: Equal
         value: system
   ```

2. Apply the VPA configuration file using `d8 k apply -f <your-config-file-name>`.

For more details on VPA resource limits configuration, see the documentation.

Module configuration example

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  version: 1
  enabled: true
  settings:
    nodeSelector:
      node-role/system: ""
    tolerations:
    - key: dedicated.deckhouse.io
      operator: Equal
      value: system
```

Additional resources

- Module `console` documentation
- Module `vertical-pod-autoscaler` documentation

Pod restart on configuration change

Deckhouse Kubernetes Platform can automatically restart Pods when certain ConfigMap or Secret resources are modified. This functionality is based on the Reloader project and is controlled via annotations added to Pod controllers (Deployment, DaemonSet, StatefulSet).

> **i** Reloader is not designed to be highly available.

The following annotations allow you to control pod restarts.

Supported annotations

Annotation	Applies to objects	Purpose	Example values
`pod-reloader.deckhouse.io/auto`	Deployment, DaemonSet, StatefulSet	Automatically restarts Pods when any related ConfigMap or Secret changes (used as a volume or environment variable)	`"true"`, `"false"`
`pod-reloader.deckhouse.io/search`	Deployment, DaemonSet, StatefulSet	Restarts only when resources annotated with `pod-reloader.deckhouse.io/match: "true"` are changed	`"true"`, `"false"`
`pod-reloader.deckhouse.io/configmap-reload`	Deployment, DaemonSet, StatefulSet	Specifies particular `ConfigMap` resources that should trigger a restart when changed	`"some-cm"`, `"some-cm1,some-cm2"`
`pod-reloader.deckhouse.io/secret-reload`	Deployment, DaemonSet, StatefulSet	Specifies particular `Secret` resources that should trigger a restart when changed	`"some-secret"`, `"some-secret1,some-secret2"`
`pod-reloader.deckhouse.io/match`	ConfigMap, Secret	Marks resources to be tracked when `pod-reloader.deckhouse.io/search: "true"` is used	`"true"`, `"false"`

> **⚠** The `pod-reloader.deckhouse.io/search` annotation must not be used together with `pod-reloader.deckhouse.io/auto: "true"`. In this case, both `pod-reloader.deckhouse.io/search` and `pod-reloader.deckhouse.io/match` annotations will be ignored. To ensure correct behavior, set `pod-reloader.deckhouse.io/auto: "false"` or remove it.
>
> The `pod-reloader.deckhouse.io/configmap-reload` and `pod-reloader.deckhouse.io/secret-reload` annotations do not work when `pod-reloader.deckhouse.io/auto: "true"` is present. To ensure correct behavior, disable `auto`.

Usage examples

Tracking all changes in all attached resources

Connected resources can be used either as environment variables or mounted as volumes.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
  annotations:
    pod-reloader.deckhouse.io/auto: "true"
spec:
  template:
    spec:
      containers:
        - name: nginx
          env:
            - name: SECRET_WORD
              valueFrom:
                secretKeyRef:
                  name: nginx-secret-value
                  key: extra
          volumeMounts:
            - name: pages
              mountPath: "/usr/share/nginx/pages"
      volumes:
        - name: pages
          configMap:
            name: nginx-pages
---
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: nginx-secret-value
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-pages
```

Tracking changes in specific resources

Indicates that only those Secrets or ConfigMaps with the annotation `pod-reloader.deckhouse.io/match: "true"` should be monitored. This is useful when a pod uses many resources, but restarts are required only upon changes to specific ones.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  annotations:
    pod-reloader.deckhouse.io/search: "true"
spec:
  template:
    spec:
      containers:
        - name: nginx
          env:
            - name: SECRET_WORD
              valueFrom:
                secretKeyRef:
                  name: nginx-secret-value
                  key: extra
---
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: nginx-secret-value
  annotations:
    pod-reloader.deckhouse.io/match: "true"
```

Tracking changes in resources from the list

Explicitly specifies a list of ConfigMaps that should trigger a pod restart when changed.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  annotations:
    pod-reloader.deckhouse.io/configmap-reload: "nginx-config,nginx-pages"
spec:
  template:
    spec:
      containers:
        - name: nginx
          volumeMounts:
            - name: pages
              mountPath: "/usr/share/nginx/pages"
            - name: config
              mountPath: "/etc/nginx/templates"
      volumes:
        - name: pages
          configMap:
            name: nginx-pages
        - name: config
          configMap:
            name: nginx-config
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-pages
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
```

Enabling or disabling pod restart

You can enable or disable the pod restart functionality in the following ways:

1. Using a `ModuleConfig` resource (e.g., `ModuleConfig/pod-reloader`). Set the `spec.enabled` field to `true` or `false`:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: pod-reloader
   spec:
     enabled: true
   ```

2. Using the `d8` command (in the `d8-system/deckhouse` pod):

   ```
   d8 system module enable pod-reloader
   ```

3. Through the Deckhouse web interface:

   - Go to the “Deckhouse → Modules” section;
   - Find the `pod-reloader` module and click on it;
   - Toggle the “Module enabled” switch.

Configuration

The pod restart mechanism works out of the box and does not require any mandatory configuration. By default, it is enabled in the Default and Managed module bundles and disabled in the Minimal bundle.

If needed, its behavior can be adjusted in the settings of the pod-reloader module (ModuleConfig `pod-reloader`).

Available parameters:

Parameter	Type	Description	Default Value
`reloadOnCreate`	boolean	Restart on ConfigMap or Secret creation, not only on modification	`true`

Parameter	Type	Description	Default Value
`nodeSelector`	object	Limits the nodes where the component can run (equivalent to `spec.nodeSelector`)	Not set
`tolerations`	array	Allows scheduling on tainted nodes (equivalent to `spec.tolerations`)	Not set

Additional resources

- Module `console` documentation
- Module `pod-reloader` documentation

Scheduler

Pod scheduling

General overview of the scheduler

The scheduler is a component responsible for assigning pods to cluster nodes based on available resources and specified rules. It selects the most suitable node for running each pod, considering many factors such as CPU and memory availability, data locality, network topology, node labels, and application-specific requirements.

The scheduler operates in multiple stages, each involving different plugins. These plugins evaluate nodes against specific criteria and help determine the optimal placement for a pod.

Main tasks of the scheduler:

- Filtering — determines which nodes meet the pod's requirements.
- Scoring — assigns scores to nodes based on various criteria (the higher the score, the better the node).
- Binding — assigns the pod to the best node based on scoring results.

Plugins may participate in one or multiple stages. For example, one plugin might be used only during filtering, while another can be involved in both filtering and scoring.

Examples of plugins:

- ImageLocality (phase: Scoring) — prefers nodes that already have the required container images. This helps reduce deployment time since images do not need to be pulled from a remote repository.
- TaintToleration (phases: Filtering, Scoring) — implements the taints and tolerations mechanism. It helps avoid placing pods on unsuitable nodes or, conversely, forces placement on specific nodes (e.g., those with dedicated resources).
- NodePorts (phase: Filtering) — checks if the node has free ports required by the pod. This is especially important for services using fixed NodePorts, as a node cannot host multiple pods using the same port.

For the complete list of plugins, refer to the Kubernetes documentation.

Pod scheduling phases

The process of assigning pods to nodes in DKP involves several key phases. The scheduler analyzes available nodes, applies various filtering and scoring criteria, and then selects the most suitable node to run the pod.

In the Filtering phase, filtering plugins are activated. These plugins evaluate all available nodes and select only those that meet certain conditions (e.g., taints, nodePorts, nodeName, unschedulable, and others).

If the nodes are distributed across different availability zones, DKP alternates between zones during selection to avoid placing all pods in a single zone. For example, if the nodes are distributed like this:

```
Zone 1: Node 1, Node 2, Node 3, Node 4
Zone 2: Node 5, Node 6
```

The selection will occur in the following order:

```
Node 1, Node 5, Node 2, Node 6, Node 3, Node 4
```

It is important to note that the scheduler does not evaluate all nodes in the cluster — only a subset is considered to optimize the scheduling process. The number of nodes evaluated depends on the size of the cluster:

- In small clusters (≤50 nodes), all nodes are evaluated.
- In medium clusters (~100 nodes), about 50% of nodes are evaluated.
- In very large clusters (~5000 nodes), only 10% of nodes are evaluated.
- The minimum threshold is 5% of nodes when there are more than 5000.

After the filtering phase, the scoring phase begins. During this stage, each node from the filtered list is scored by various scoring plugins that analyze its characteristics and how well it matches the pod's requirements.

Each plugin evaluates a node based on its specific criteria and assigns it a score. These scores are then aggregated to produce an overall ranking.

Key factors considered during scoring include:

- Node load (`pod capacity`) — available CPU and memory are taken into account. Nodes with more free resources receive higher scores.
- Container image locality (ImageLocality) — nodes that already have the required container images are preferred, as this speeds up pod deployment.
- Affinity and anti-affinity rules — nodes that satisfy required placement (or avoidance) rules based on other pods get higher priority.
- Storage availability (`volume provisioning`) — nodes with the necessary PersistentVolumes receive a higher score.

After all scores are summed, the node with the highest total score is selected for pod placement.

If multiple nodes receive the same highest score, the selection is made randomly to avoid inefficient and predictable load distribution.

How to modify or extend scheduler logic

The scheduler in DKP can be flexibly configured and extended using custom plugins. This allows its behavior to be adapted to specific cluster requirements, such as considering custom metrics, special workload distribution rules, or node prioritization.

Each such plugin is implemented as a webhook and must meet the following requirements:

- TLS usage – The plugin must be accessed over a secure TLS connection.
- Availability – The plugin must be deployed as a service inside the cluster and accessible via HTTP(S).
- Support for standard verbs – The plugin must support standard scheduler operations:
  - `filterVerb = filter` – participates in node filtering before pod placement.
  - `prioritizeVerb = prioritize` – participates in node scoring during the Scoring phase.
- Node cache capability – It is expected that all extender plugins support caching node data (`nodeCacheCapable: true`) to improve performance.

To connect an extender plugin, use the KubeSchedulerWebhookConfiguration resource. This resource defines the configuration for an external webhook used by the kube-scheduler and enables more advanced scheduling conditions in the cluster, such as:

- Placing application pods closer to data storage nodes,
- Prioritizing nodes based on their current state (e.g., network load or storage subsystem health),
- Segmenting nodes into zones, etc.

Example of connecting an external scheduler plugin via webhook:

```
apiVersion: deckhouse.io/v1alpha1
kind: KubeSchedulerWebhookConfiguration
metadata:
  name: sds-replicated-volume
webhooks:
- weight: 5
  failurePolicy: Ignore
  clientConfig:
    service:
      name: scheduler
      namespace: d8-sds-replicated-volume
      port: 8080
      path: /scheduler
    caBundle: ABCD=
  timeoutSeconds: 5
```

✖ If the `failurePolicy: Fail` parameter is used, any failure in the webhook will cause the scheduler to stop functioning entirely, preventing new pods from being scheduled. It is highly recommended to thoroughly test any custom scheduler plugins before deploying them in a production environment.

Additional pod placement mechanisms

Deckhouse Kubernetes Platform provides flexible mechanisms for managing pod placement within a cluster. These mechanisms help optimize load balancing, improve fault tolerance, and separate system and user workloads.

Main placement configuration options:

- Using node labels (`NodeGroup.spec.nodeTemplate.labels`).
  Allows explicitly targeting specific nodes for pods using `spec.nodeSelector` or `spec.affinity.nodeAffinity`.
- Configuring taints and tolerations.
  Enables limiting pod scheduling to specific nodes, preventing them from being placed on unsuitable hosts.
- Using custom toleration keys (`settings.modules.placement.customTolerationKeys`).
  Allows control over the scheduling of critical Deckhouse components (e.g., CNI and CSI) on dedicated nodes.

1. Using `nodeSelector` — allows you to explicitly specify which nodes pods should be scheduled on. This is done by labeling the desired nodes and referencing those labels in the pod's `spec.nodeSelector`. Example:

   Suppose you have a node named `kube-system-1` that is intended for monitoring services. Label the node:

   ```
   d8 k label node kube-system-1 node-role/monitoring=""
   ```

Now, to deploy pods only on this node, add a `nodeSelector` to the Deployment:

```
nodeSelector:
 node-role/monitoring: ""
```

2. Using taints and tolerations — unlike `nodeSelector`, taints are used to prevent pods from being scheduled on a node unless they have a matching toleration. This ensures that only specific services run on a given node. Example:

   Suppose you have a node named `kube-frontend-1` that is dedicated exclusively to Ingress controllers. Apply a taint to the node:

   ```
   d8 k taint node kube-frontend-1 node-role/frontend="":NoExecute
   ```

   Now, only pods with the appropriate toleration will be allowed to run on this node:

   ```
   tolerations:
   - effect: NoExecute
     key: node-role/frontend
   ```

   This mechanism prevents unintended pods from being scheduled on nodes that are meant for specific purposes.

3. Using `customTolerationKeys` — Deckhouse supports the `customTolerationKeys` mechanism, which explicitly defines allowed toleration keys. This is useful when you need to run system services (such as CNI, CSI, etc.) on specific nodes. Example:

   ```
   customTolerationKeys:
   - dedicated.example.com
   - node-dedicated.example.com/master
   ```

   This mechanism allows:

   - Divide nodes into zones — for example, dedicating some for Ingress controllers and others for system services (Prometheus, VPN, CoreDNS).
   - Isolate critical applications from system components, avoiding resource contention.

Scheduler profiles

The scheduler supports multiple profiles that define different strategies for distributing pods across nodes in the cluster. Depending on the selected profile, pods will be scheduled with different load balancing logic.

- `default-scheduler` — the default profile that attempts to evenly distribute pods across nodes, preferring less loaded ones.
- `high-node-utilization` — a profile that places pods on more heavily loaded nodes. This can be useful in scenarios where workload consolidation is needed, allowing underutilized nodes to be shut down or repurposed.

To select a scheduler profile, specify it in the `spec.schedulerName` field of the Pod manifest. Example:

```
apiVersion: v1
kind: Pod
metadata:
  name: scheduler-example
  labels:
    name: scheduler-example
spec:
  schedulerName: high-node-utilization
  containers:
  - name: example-pod
    image: registry.k8s.io/pause:2.0
```

Pod redistribution

DKP analyzes the state of the cluster every 15 minutes and evicts pods that match the conditions described in active scheduling strategies. Evicted pods go through the standard scheduling process again, taking the current cluster state into account. This mechanism allows workloads to be redistributed based on the selected strategy and frees up resources on certain nodes when necessary.

Considering pod Priority Classes

DKP defines a set of priority classes that determine the importance of pods and the order in which they are evicted during workload redistribution.

If the cluster lacks resources, lower-priority pods may be evicted in favor of higher-priority ones. This ensures that critical services remain operational even when nodes are overloaded.

How DKP considers pod priority

Deckhouse Kubernetes Platform uses the pod priority mechanism to determine which pods should be evicted when resources are insufficient. The higher the pod’s priority, the lower the chance it will be evicted during redistribution.

This mechanism is controlled by the `descheduler` module, where you can define a priority threshold using the `spec.priorityClassThreshold` parameter. This threshold limits eviction to only those pods with a priority below the specified value.

You can set the threshold in two ways:

- By class name: Use `priorityClassThreshold.name` to evict only pods with a priority lower than the specified priority class.
- By numeric value: Use `priorityClassThreshold.value` to evict pods with a priority lower than the specified integer value.

Example:

```
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: custom
spec:
  priorityClassThreshold:
    name: high-priority
```

Which pods are not evicted

- DKP does not evict a pod in the following cases:
  - The pod is in the `d8-*` or `kube-system` namespace;
  - The pod has the `priorityClassName` set to `system-cluster-critical` or `system-node-critical`;
  - The pod is using local storage;
  - The pod is managed by a DaemonSet;
  - Evicting the pod would violate a Pod Disruption Budget (PDB);
  - There are no available nodes to reschedule the evicted pod.

If multiple pods match the eviction criteria, DKP applies additional logic:

1. Pods with the lowest priority (`BestEffort`) are evicted first;
2. Then `Burstable` pods are considered;
3. `Guaranteed` pods are evicted last, and only if absolutely necessary.

DKP provides fine-grained control over which pods and nodes are subject to eviction:

- `spec.podLabelSelector` — limits evicted pods by label;
- `spec.namespaceLabelSelector` — filters namespaces whose pods can be considered for eviction;
- `spec.nodeLabelSelector` — selects target nodes by label.

Each of these fields supports standard `matchExpressions` and `matchLabels` syntax, allowing you to use operators like `In`, `NotIn`, `Exists`, and `DoesNotExist` with desired label values.

How to enable or disable pod eviction

To enable the pod redistribution feature, you need to enable the `descheduler` module.

You can do this in one of the following ways:

1. Using a ModuleConfig resource (e.g., `ModuleConfig/descheduler`). Set the `spec.enabled` field to `true` or `false`:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: descheduler
   spec:
     enabled: true   # or false to disable
   ```

2. Using the `d8` command (in the `d8-system/deckhouse` pod):

   ```
   d8 system module enable descheduler
   # or to disable:
   d8 system module disable descheduler
   ```

3. Through the Deckhouse web interface:
   - Go to the “Deckhouse → Modules” section;
   - Find the `descheduler` module and click on it;
   - Toggle the “Module enabled” switch.

The module does not require mandatory configuration. You can enable it without additional settings — it will work with the default values.

Descheduling strategies

The `spec.strategies` parameter lists the strategies you want to enable or configure. Each strategy has an `enabled` flag (default is `false`).

Below is a list of the main strategies available in DKP.

HighNodeUtilization — concentrates workloads on fewer nodes by evicting pods from underutilized nodes so they can be rescheduled elsewhere. Requirements:

- Specific `descheduler` module configuration — `MostAllocated`;
- (Optional) cluster autoscaling must be enabled — to allow unused nodes to be shut down.

This strategy is enabled using the `spec.strategies.highNodeUtilization.enabled` parameter.

The `thresholds` parameter defines the resource usage levels below which a node is considered underutilized. If usage (CPU, memory, etc.) falls below *all* threshold values, the node is deemed underutilized, and DKP will attempt to evict pods from it.

Example:

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: high-node-utilization
spec:
  strategies:
    highNodeUtilization:
      enabled: true
      thresholds:
        cpu: 50
        memory: 50
```

If any of these resource types is not specified, all its thresholds default to 100 to avoid nodes going from underutilized to overutilized For example, If the memory threshold is not specified in thresholds, it defaults to 100.

```
thresholds:
  cpu: 50
```

> **i** In GKE (Google Kubernetes Engine), it is not possible to configure `MostAllocated` by default, but you can use the `optimize-utilization` strategy instead.

LowNodeUtilization — more evenly distributes workloads across nodes. This strategy identifies underutilized nodes and evicts pods from overutilized ones, assuming that evicted pods will be rescheduled on the underutilized nodes.

Enable the strategy via `spec.strategies.lowNodeUtilization.enabled`.

- An **underutilized** node is one where all resources are below the thresholds set in `strategies.lowNodeUtilization.thresholds`.
- An **overutilized** node exceeds at least one of the values in `strategies.lowNodeUtilization.targetThresholds`.

Nodes with resource usage between `thresholds` and `targetThresholds` are considered optimally utilized and will not have pods evicted.

Example:

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: low-node-utilization
spec:
  strategies:
    lowNodeUtilization:
      enabled: true
      thresholds:
        cpu: 20
      targetThresholds:
        cpu: 50
```

RemoveDuplicates — prevents multiple pods from the same controller (ReplicaSet/ReplicationController/StatefulSet/Job) from running on the same node simultaneously (DaemonSets are excluded). If, for any reason, 2 or more pods from the same controller end up on a single node, this strategy will evict the “extra” pods to redistribute them more evenly across the cluster.

This situation can occur after temporary node failures — when pods are rescheduled to other nodes, and once the original node is back online, the pods remain concentrated on the fallback node.

Enable the strategy using the `strategies.removeDuplicates.enabled` parameter.

Example:

```
spec:
  strategies:
    removeDuplicates:
      enabled: true
```

RemovePodsHavingTooManyRestarts — evicts pods having too many restarts from nodes. Pods become eviction candidates when the total number of restarts across all containers, including init containers, exceeds the `podRestartThreshold` threshold. This strategy is useful for evicting pods in the `CrashLoopBackOff` state or with repeated failures, as well as for freeing up resources and allowing fresh pods to be scheduled on potentially healthier nodes.

Enable the strategy using the `strategies.removePodsHavingTooManyRestarts.enabled` parameter.

Example:

```
spec:
  strategies:
    removePodsHavingTooManyRestarts:
      enabled: true
```

RemovePodsViolatingInterPodAntiAffinity — evicts any pods that violate `inter-pod affinity/anti-affinity` rules. For example, if `pod2` and `pod3` have an `anti-affinity` rule against `pod1`, but all three end up on the same node, this strategy will evict `pod1` so that `pod2` and `pod3` can continue operating according to their scheduling constraints.

Enable the strategy using the `strategies.removePodsViolatingInterPodAntiAffinity.enabled` parameter.

Example:

```
spec:
  strategies:
    removePodsViolatingInterPodAntiAffinity:
      enabled: true
```

RemovePodsViolatingNodeAffinity — this strategy evicts pods that no longer comply with their defined `node affinity` rules. `Node affinity` defines which nodes are suitable for a pod at scheduling time:

1. `requiredDuringSchedulingIgnoredDuringExecution`:
   - A pod with this rule must be scheduled only on a node that satisfies the specified conditions (e.g., a specific label).
   - If the node later no longer satisfies the condition (e.g., the label is removed), and there is another suitable node in the cluster, DKP will evict the pod so it can be rescheduled on a valid node.
2. `preferredDuringSchedulingIgnoredDuringExecution`:
   - A pod with this preference may run on a node that doesn't fully match the affinity, if no better option is available.
   - If a more suitable node appears later in the cluster, DKP may evict the pod so it can be restarted under better placement conditions.

This strategy helps ensure pods continue to align with their node affinity rules and do not remain in suboptimal locations when better options become available.

Enable the strategy using the `spec.strategies.removePodsViolatingNodeAffinity.enabled` parameter.

Example:

```
spec:
  strategies:
    removePodsViolatingNodeAffinity:
      enabled: true
      nodeAffinityType:
        - requiredDuringSchedulingIgnoredDuringExecution
        - preferredDuringSchedulingIgnoredDuringExecution
```

RemovePodsViolatingTopologySpreadConstraint — ensures that pods violating topology spread constraints are evicted from nodes. It evicts the minimum number of pods required to balance topology domains to within each constraint's `maxSkew`. This is useful for rebalancing pods across zones after a zone outage recovery.

Enable the strategy using the `strategies.removePodsViolatingTopologySpreadConstraint.enabled` parameter.

Example:

```
spec:
  strategies:
    removePodsViolatingTopologySpreadConstraint:
      enabled: true
```

Additional resources

- Global parameters
- Module `console` documentation

- Module `control-plane-manager` custom resources
- Module `descheduler` documentation
- Module `node-manager` custom resources

Priority classes

Pod priorities (Priority Classes)

Available Priority Classes

DKP creates a set of predefined PriorityClass objects in the cluster and assigns them to Deckhouse components and user applications. The Kubernetes scheduler takes pod priority into account when allocating resources: if there are not enough resources, pods with lower priorities will be evicted first.

For example, if pods are assigned `priorityClassName: production-low` and the cluster runs out of resources, the scheduler will first evict pods with `priorityClassName: develop`, then `cluster-low`, and so on. Therefore, when selecting a priority class, you should consider the type of application and the environment in which it runs. If a pod has no specified priority, the scheduler treats it as the lowest.

✖ Do not use the `system-node-critical`, `system-cluster-critical`, `cluster-medium`, or `cluster-low` priority classes, as they are reserved for critical cluster components.

Below is a table of available priority classes.

Priority Class	Description	Value
system-node-critical	Cluster components that must be present on the node. Also fully protected from eviction by kubelet. **Examples:** node-exporter, csi, and others.	2000001000
system-cluster-critical	Cluster components without which the cluster cannot function properly. This PriorityClass must be assigned to MutatingWebhooks and Extension API servers. Also fully protected from eviction by kubelet. **Examples:** kube-dns, kube-proxy, cni-flannel, cni-cilium, and others.	2000000000
production-high	Stateful applications in production environments whose absence leads to complete service unavailability or data loss. **Examples:** PostgreSQL, Memcached, Redis, MongoDB, and others.	9000
cluster-medium	Cluster components responsible for monitoring (alerts, diagnostics) and autoscaling. Without monitoring it's impossible to assess incidents; without autoscaling, applications may lack necessary resources. **Examples:** deckhouse, node-local-dns, grafana, upmeter, and others.	7000
production-medium	Main stateless applications in production that serve end-users.	6000
deployment-machinery	Cluster components used for build and deploy processes.	5000
production-low	Production applications (cron jobs, admin panels, batch processes) that can be temporarily unavailable. If batch/cron jobs cannot be interrupted, they should be classified as `production-medium`.	4000
staging	Staging environments for applications.	3000
cluster-low	Non-critical but desirable cluster components. **Examples:** dashboard, cert-manager, prometheus, and others.	2000
develop (default)	Development environments for applications. Default priority class if none is specified.	1000
standby	Not intended for applications. Used internally to reserve nodes.	-1

Network

Unlike basic Kubernetes installations, Deckhouse Kubernetes Platform (DKP) handles network configuration through modules and custom resources, which simplifies and standardizes the setup process.

The platform supports:

- Management of incoming and outgoing traffic.
- Configuration of inter-cluster communication.
- Management of network policies and routing.
- DNS management within the Kubernetes cluster.

Configuring and managing incoming traffic

Incoming traffic is managed using Ingress controllers. These controllers route user requests to the appropriate applications and services based on rules defined in Ingress resources. In DKP, traffic can be managed at both the network level (NLB – Network Load Balancer) and the application level (ALB – Application Load Balancer).

Managing outgoing traffic

DKP administrators can configure and manage outgoing traffic to ensure correct processing and routing of all outgoing data.

Configuring outgoing traffic enables connection control and filtering, which is especially important for secure integrations with external APIs and services. This can be achieved by defining policies that allow or block specific traffic flows.

Outgoing traffic management is implemented using the Egress Gateway feature (provided by the `cni-cilium` module) as well as Istio tools (via the `istio` module).

Internal network configuration

DKP provides a wide range of options for managing the internal network. Administrators can configure pod-to-pod and pod-to-node communication, as well as traffic encryption using various technologies and tools.

Configuring inter-cluster communication

DKP supports two approaches to organizing communication between independent and codependent clusters:

- Multicluster — combines clusters to share resources, balance loads, and improve fault tolerance. Ideal for distributed teams and infrastructure spread across regions.
- Federation — centralized management of multiple independent clusters with unified policies and synchronized configurations. Effective for large organizations and complex systems.

For more information on configuring inter-cluster communication, refer to Inter-cluster communication.

Network policies

Network policies in DKP define rules that regulate traffic flow between pods, nodes, namespaces, and external systems. Network policies ensure pod isolation, protect against internal cluster attacks, and provide control over access to external services as well as incoming and outgoing connections.

Additional resources

- Module `cni-cilium` documentation
- Module `istio` documentation

Internal network configuration

In Deckhouse Kubernetes Platform, networking is configured using CNI plugins. The recommended option is Cilium, which suits most use cases. Other supported CNIs include Flannel and Simple Bridge, typically used with cloud providers.

Network parameters are specified during the deployment of the DKP cluster:

- Address ranges for Pods and Services are defined.
- The network operation mode for Cilium is selected (this can be changed later).

Internal network operation modes

The operation mode is set via the `tunnelMode` parameter in the `cni-cilium` module settings. Two modes are supported:

- Classic
- VXLAN tunnel

The appropriate mode depends on the cluster network setup:

- If all nodes are on the same network, either mode can be used.
- If nodes are on different networks, using VXLAN is recommended to create a virtual isolated network over the existing one. This allows communication across different network segments.

Performance differences between the modes are minimal. The table below outlines their characteristics:

Parameter	Classic mode	VXLAN tunnel mode
Network type	For nodes in the same network	Suitable for nodes in different networks

Parameter	Classic mode	VXLAN tunnel mode
Traffic isolation	Not provided	Provided
Infrastructure usage	Direct routing	Virtual network over existing infrastructure
Changes to MTU	No changes	MTU is reduced due to encapsulation
Additional routing	Intermediate hop is added	Not required

Mode configuration

The internal network mode is configured via the ModuleConfig resource named `cni-cilium`.

Example configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  version: 1
  enabled: true
  settings:
    tunnelMode: VXLAN
```

For more details on configuring the `cni-cilium` module, refer to the module documentation.

⚠ After changing the operation mode, reboot all nodes. Otherwise, Pod availability issues may occur.

Services

Services in DKP are implemented either using the selected CNI plugin or through `kube-proxy`.

They provide access to groups of Pods performing the same function and balance traffic between them. Each Service object defines:

- A logical group of endpoints (Pods).
- A routing method for reaching them.

Supported service types

- **ClusterIP**: Creates an internal IP address selected from the Service network. All traffic sent to this IP is forwarded to the Pods specified in the selector. This ensures load balancing.

  You can also disable IP assignment in this service type. In that case, the service will not be assigned an IP address from the service network, and it will resolve to the Pod IPs. Load balancing will be done using DNS (round-robin).

- **NodePort**: Opens the specified port on each cluster node and forwards incoming traffic to the Pods. For security reasons, DKP only listens on the node's internal IP. This behavior can be overridden by adding the annotation `node.deckhouse.io/nodeport-bind-internal-ip: "false"` to a node group.
- **LoadBalancer**: Created in the cloud provider where DKP is deployed and accepts external traffic. In bare-metal clusters, similar functionality is implemented using the `metalLb` module.
- **ExternalName**: A DNS record for accessing the service (acting as a CNAME).

Configuring the service network with kube-proxy

When using Flannel or Simple Bridge, the `kube-proxy` module is required for services to function. It replaces the default kubeadm components (DaemonSet, ConfigMap, RBAC) with its own configuration.

By default, for security reasons, when using services of NodePort type, connections are only accepted on the node's InternalIP. This behavior can be changed by adding the following annotation to the node: `node.deckhouse.io/nodeport-bind-internal-ip: "false"`.

Example annotation for a NodeGroup:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: myng
spec:
  nodeTemplate:
    annotations:
      node.deckhouse.io/nodeport-bind-internal-ip: "false"
```

> ⚠ After adding, removing, or changing this annotation, you must manually restart all kube-proxy Pods.
>
> When the `cni-cilium` module is enabled, the `kube-proxy` module is automatically disabled.

Additional resources

- Module `cni-cilium` documentation
- Module `metallb` parameters

Encrypting traffic between pods

To encrypt traffic between pods in Deckhouse Kubernetes Platform (DKP), you can use mTLS provided by Istio (via the `istio` module).

mTLS (mutual TLS) enables mutual authentication between services using TLS certificates: for all outgoing requests, the server certificate is verified; for incoming requests, the client certificate is checked. Once verification is successful, the sidecar proxy identifies the remote node and can use this information for authorization or other application-level purposes.

Each service receives a unique identifier in the format `<TrustDomain>/ns/<Namespace>/sa/<ServiceAccount>`.

Where:

- `TrustDomain` is the cluster domain.
- `ServiceAccount` is the account name (either `default` or a custom one).

This identifier is used as the verifiable name in TLS certificates.

The settings can be overridden at the namespace level.

mTLS configuration example

1. Enable the `istio` module:

   ```
   d8 k create -f -<<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: istio
   spec:
     version: 2
     enabled: true
   EOF
   ```

2. Create a namespace and add labels:

   ```
   d8 k create namespace test-istio-mtls
   d8 k label namespace test-istio-mtls istio-injection=enabled
   d8 k label namespace test-istio-mtls security.deckhouse.io/pod-policy=privileged
   ```

3. Apply a policy that defines the mTLS mode. Available modes:

   - `PERMISSIVE` (default): Allows encrypted traffic between pods, but also permits unencrypted traffic.
   - `STRICT`: Only encrypted traffic is allowed.

   Example of applying `PERMISSIVE` mode:

   ```
   d8 k -n test-istio-mtls create -f -<<EOF
   apiVersion: security.istio.io/v1beta1
   kind: PeerAuthentication
   metadata:
     name: default
   spec:
     mtls:
       mode: PERMISSIVE
   EOF
   ```

4. Create a Deployment:

   - If using a public registry:

     ```
     d8 k -n test-istio-mtls create deployment webserver --image=docker.io/library/nginx:1.26-alpine --port 80
     ```

   - If using an all-in-one image (replace with your registry address):

     ```
     d8 k -n test-istio-mtls create deployment webserver --image=registry.company.network/localrepo/all-in-one-image:0.1 --port 80 -- /bin/sh -c 'nginx -g "daemon off;"'
     ```

5. Expose the Deployment:

   ```
   d8 k -n test-istio-mtls expose deployment webserver --port=80
   ```

6. Get the Pod IP or the associated node name. It will be required to run `tcpdump` on the respective node:

   ```
   d8 k -n test-istio-mtls get pods -l app=webserver -o wide
   ```

   Example output:

   ```
   NAME                         READY   STATUS     RESTARTS   AGE   IP              NODE
   NOMINATED NODE    READINESS GATES
   webserver-76d6c9b8c-9mdtb    2/2     Running    0          48m   10.111.1.122    test-worker-
   e36e4712-5948b-sp9t8         <none>             <none>
   ```

7. Access the node via SSH as root and run `tcpdump`:

   ```
   tcpdump -A -v -i any host 10.111.1.122 and port 80
   ```

8. Create a Deployment client in the same namespace so that it is part of the mesh network:

   - If using a public registry image:

     ```
     d8 k -n test-istio-mtls create deployment client --image=docker.io/library/alpine:3.21 -- /bin/sh -c "sleep
     infinity"
     ```

   - If using an all-in-one image (replace with your registry address):

     ```
     d8 k -n test-istio-mtls create deployment client --image=registry.company.network/localrepo/all-in-one-
     image:0.1 -- /bin/sh -c "sleep infinity"
     ```

9. After the Pod is created, make a request to the `webserver` service:

   ```
   d8 k -n test-istio-mtls exec -ti deployments/client -- wget -S --spider --timeout 1 webserver`
   ```

   The `tcpdump` output will show only encrypted traffic:

   ```
   10.111.91.68.http > 10.111.91.167.54204: Flags [P.], cksum 0xd7c7 (incorrect -> 0xe83a), seq 1:3033, ack 518, win
   134, options [nop,nop,TS val 3160034944 ecr 2239691194], length 3032: HTTP
   E....]@.@.T.
   o[D
   o[..P..'...;.i8..........
   .ZN..~......z...v...T......d.4p
   ..6j..V..G...0 d..U h...e....W.`.....4....;.......T.......3.$... [..%ex.....i<...*.
   [..C...jC....p.+..............Nt ................
   {..P%.+.,L......A........B..7}.v../...rg.o.)5......E.X;K8..%,..yb.^f...^+..Ble.j..w3 S......}..
   |3...+.=...;H..#...Y...3h.....:.9.w.A.4X..g...|S;.`<....%.y..4.....D.m...../6.7[.......!+.J....
   ....2._r.D.C>`.3A..... ..H.^1...P1N....)..%...
   r.=..s.FB..i^..^....4{;...ED_.\...g`l,q........U*0.#..(......).h..|sc...
   .c(..kd....5C@..5.Z.L..C2
   @.7>..y%....R.5........j9........!........H.a....b..].0..~..0x7..+N...o.6........
   (%7:.Xr..N......'.O.a.j&v..Ba.....t..q.
   =U.t.fU.].......2g....Eat.D.3n........*G.N..!LY.......n)........_jL..9RD..gT.lX..p&..=.d..Tq%....qF`.....'..|..$!g..
   j.d. N%tb^...@- ..`...S:..D.....y...ckA.;}Y,.....X..6...=[,..|AD..._}......W.. ..u3G...<.
   ..&.0j.,.'!.#
   ....w..bx.............}U.;..y....J.K..fQ.#]..3.V..=.d._.'....q.;!.9.N......n.7.Zi.>....@...].u.A}.;.....c
   ..s......d.*=..G..9......Nt@....v..s.>.
   ...h....Cm.Z.......n......L........-
   _.......r...%Z.....h...........`..8A....yt.t..2d....oH?.1.O&.J....F..b.OV.............E1H..%~..
   2.H..{.I...=.I.*..2y1p0h..........P.....@r....vk.!".......{..`.3..
   <,.r&L.....M...t...;.z...Q...1+.,........:._L.V.....e.. ..!]\..6*}....vT.A>.....A0.....
   {.?}AH.+R..g.[=?.X...|94....S+!..e...*..M`f...o..b.K#5.....@...5.......o.(.
   ...
   ```

10. Create a namespace without the `istio-injection=enabled` label and make a request to the `webserver.test-istio-mtls` service (a request is made from a Pod that is not part of the mesh network):

    ```
    d8 k create namespace test-istio-mtls-without-injection
    ```

11. Add a Deployment:

    - If using a public registry image:

```
d8 k -n test-istio-mtls-without-injection create deployment alpine --image=docker.io/library/alpine:3.21 --
/bin/sh -c "sleep infinity"
```

- If using an all-in-one image (replace with your registry address):

```
d8 k -n test-istio-mtls create deployment client --image=registry.company.network/localrepo/all-in-one-
image:0.1 -- /bin/sh -c "sleep infinity"
```

12. After the Pod is created, make a request to the `webserver.test-istio-mtls` service:

```
d8 k -n test-istio-mtls-without-injection exec -ti deployments/alpine -- wget -S --spider --timeout 1
webserver.test-istio-mtls`.
```

The `tcpdump` output will show only unencrypted (plain text) requests and responses:

```
10.111.91.211.54424 > 10.111.91.68.http: Flags [P.], cksum 0xcc77 (incorrect -> 0xee57), seq 1:93, ack 1, win 128,
options [nop,nop,TS val 305286128 ecr 3834507600], length 92: HTTP, length: 92
GET / HTTP/1.1
Host: webserver.test-istio-mtls
User-Agent: Wget
Accept: */*
Connection: close

E...y.@.?...
o[.
o[D...P....<Nb......w.....
.2K....PGET / HTTP/1.1
Host: webserver.test-istio-mtls
User-Agent: Wget
Accept: */*
Connection: close

09:14:20.960302 lxc4f4a182c887c In  IP (tos 0x0, ttl 64, id 33396, offset 0, flags [DF], proto TCP (6), length 52)
10.111.91.68.http > 10.111.91.211.54424: Flags [.], cksum 0xcc1b (incorrect -> 0xa850), ack 93, win 128, options
[nop,nop,TS val 3834507601 ecr 305286128], length 0
E..4.t@.@..Z
o[D
o[..P..<Nb.............
...Q.2K.
09:14:20.964496 lxc4f4a182c887c In  IP (tos 0x0, ttl 64, id 33397, offset 0, flags [DF], proto TCP (6), length 1003)
10.111.91.68.http > 10.111.91.211.54424: Flags [P.], cksum 0xcfd2 (incorrect -> 0x44eb), seq 1:952, ack 93, win 128,
options [nop,nop,TS val 3834507605 ecr 305286128], length 951: HTTP, length: 951
HTTP/1.1 200 OK
server: istio-envoy
date: Fri, 24 Jan 2025 09:14:20 GMT
content-type: text/html
content-length: 615
last-modified: Wed, 14 Aug 2024 05:42:07 GMT
etag: "66bc43af-267"
accept-ranges: bytes
x-envoy-upstream-service-time: 0
connection: close
x-envoy-decorator-operation: webserver.test-istio-mtls.svc.cluster.local:80/*

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Additional resources

- nginx.com

- nginx.org
- Module `istio` documentation

Configuring cluster network stack visualization

To visualize network interactions in the cluster between pods, services, and external resources, as well as to monitor activity and identify network issues, use the `cilium-hubble` module.

Once you enable the module, the network stack visualization becomes available through the Hubble web interface.

Additional resources

- Module `cilium-hubble` documentation

Switching CNI from Flannel or Simple bridge to Cilium

> **i** This guide is applicable to Deckhouse Kubernetes Platform version 1.75 and earlier.
>
> For DKP version 1.76 and later, use the Switching CNI in the cluster guide.

> ⚠ The migration requires a **full cluster downtime**: all pods and nodes will be restarted. Estimated time to complete is **about 1 hour**.

To perform the migration, follow these steps:

1. Enable the `cni-cilium` module:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: cni-cilium
   spec:
     version: 1
     settings:
       tunnelMode: VXLAN
     enabled: true
   ```

2. Wait for the Cilium components to start:

   ```
   d8 k -n d8-cni-cilium get pods
   ```

   Example output:

   ```
   NAME                      READY STATUS  RESTARTS    AGE
   agent-5zzfv               2/2   Running 5 (23m ago) 26m
   agent-gqb2b               2/2   Running 5 (23m ago) 26m
   agent-wtv4p               2/2   Running 5 (23m ago) 26m
   operator-856d69fd49-mlglv 2/2   Running 0           26m
   safe-agent-updater-26qpk  3/3   Running 0           26m
   safe-agent-updater-qlbrh  3/3   Running 0           26m
   safe-agent-updater-wjjr5  3/3   Running 0           26m
   ```

   If the Cilium agent pods do not transition to the `Ready` state:

   - Save the `d8-kube-proxy` manifest:

     ```
     d8 k -n kube-system get ds d8-kube-proxy -oyaml > d8-kube-proxy.yaml
     ```

   - Run the recovery actions:

     ```
     d8 k delete validatingadmissionpolicies.admissionregistration.k8s.io label-objects.deckhouse.io
     d8 k -n kube-system delete ds d8-kube-proxy
     d8 k -n d8-cni-cilium delete po -l app=agent
     ```

   - If Cilium startup still takes more than 5 minutes, restart `kube-proxy`:

     ```
     d8 k -n kube-system delete pod -l k8s-app=kube-proxy
     ```

3. Save the DaemonSet manifest:

```
d8 k -n d8-cni-simple-bridge get ds simple-bridge -o yaml > simple-bridge.yaml
# or
d8 k -n d8-cni-flannel get ds flannel -o yaml > flannel.yaml
```

4. Disable `cni-simple-bridge` or `cni-flannel`. Perform this step only after the Cilium pods have reached the `Ready` state:

```
d8 k -n d8-system exec -it svc/deckhouse-leader -- deckhouse-controller module disable cni-simple-bridge
# or
d8 k -n d8-system exec -it svc/deckhouse-leader -- deckhouse-controller module disable cni-flannel
```

5. Delete the `validating webhook` (if present):

```
d8 k delete validatingwebhookconfigurations.admissionregistration.k8s.io d8-deckhouse-validating-webhook-handler-hooks
```

6. Delete the namespace of the old CNI (if it was not removed automatically):

```
d8 k delete ns d8-cni-simple-bridge
# or
d8 k delete ns d8-cni-flannel
```

7. (Optional) Clean up artifacts of the old CNI. If required, remove the interface and flush iptables rules:

```
ip link delete cni0

iptables -t filter -F
iptables -t filter -X
iptables -t nat -F
iptables -t nat -X
iptables -t mangle -F
iptables -t mangle -X
iptables -t raw -F
iptables -t raw -X
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
```

8. Restart all pods:

```
d8 k delete pods -A --all --wait=false
```

9. Configure internal LoadBalancers (if used). For all TargetGroups that belong to the internal load balancer, set:

```
Preserve client IP addresses = Off
```

10. Reboot the cluster nodes one by one, including the master nodes.

Incoming traffic balancing

This section describes the approaches to balancing incoming traffic in Deckhouse Kubernetes Platform (DKP):

- NLB (Network Load Balancer) — operates at the network level, routing traffic based on IP addresses and ports without inspecting request contents.
- ALB (Application Load Balancer) — operates at the application level, analyzing HTTP(S) headers, paths, and domains. It supports SSL termination and content-based routing.

Network-level load balancing (NLB)

NLB-based load balancing can be implemented in two ways:

- Using an external load balancer provided by a cloud provider.
- Using the built-in MetalLB balancer, which works in both cloud and bare-metal clusters.

Application-level load balancing (ALB)

For application-level traffic balancing, DKP provides following solutions:

- Ingress NGINX Controller (via the `ingress-nginx` module).
- Kubernetes Gateway API (`alb` module),
- Istio (via the `istio` module).

Additional resources

- Module `alb` documentation
- Module `ingress-nginx` documentation
- Module `istio` documentation

Balancing with MetalLB

The `metallb` module implements support for LoadBalancer-type services in Deckhouse Kubernetes Platform (DKP) clusters. It is suitable for both bare-metal clusters and cloud environments where built-in load balancers by providers are unavailable.

Two operating modes are supported:

- **Layer 2**: An enhanced version of the standard L2 mode in MetalLB, allowing multiple public addresses to be used for services.
- **BGP**: Fully based on the MetalLB solution and available only in DKP Enterprise Edition.

Layer 2 mode

How it works

In Layer 2 mode, one or more cluster nodes receive traffic for a service from the public network. From the network layer's perspective, each of these nodes has multiple IP addresses assigned to its network interface. This is implemented by the module responding to ARP requests (for IPv4) and NDP requests (for IPv6).

The main advantage of this mode is universality. It works in any Ethernet network without requiring specialized hardware.

Advantages over classic MetalLB

In classic MetalLB (L2 mode), when creating a LoadBalancer service, load balancing is achieved by having a single node in the cluster reply to ARP requests for the public IP. This means:

- Only one node handles all incoming traffic for the given IP at a time.
- The leader node becomes a bottleneck with no horizontal scaling.
- If the leader node fails, all active connections are dropped during the switchover to the new node.

The `metallb` module removes these limitations. It provides the MetalLoadBalancerClass resource, which lets you:

- Bind a group of nodes to an IP address pool using `nodeSelector`.
- Create a standard LoadBalancer Service and specify the name of the target MetalLoadBalancerClass.
- Define the number of IP addresses for L2 announcement via an annotation.

This approach means:

- The application receives multiple public IPs, which must be added as A records in DNS.
- To scale, add more load balancer nodes. Associated Services will be created automatically — only adding the IPs to the application domain's A records is required.
- If one balancer fails, only part of the traffic is rerouted, without a complete connection drop.

Behavior comparison

Feature	Classic MetalLB (L2)	New module with MetalLoadBalancerClass
Traffic handling	Single node (leader)	Multiple nodes
Scalability	No	Yes
Fault tolerance	All connections dropped	Part of the traffic is rerouted smoothly
Number of public IPs	One	Multiple (configurable)
DNS configuration	One A record	Multiple A records

Example of using MetalLB in L2 LoadBalancer mode

1. Enable the `metallb` module:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: metallb
   spec:
     enabled: true
     version: 2
   ```

2. Deploy the application to expose:

   ```
   d8 k create deploy nginx --image=nginx
   ```

3. Create a MetalLoadBalancerClass resource:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: ingress
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  isDefault: false
  nodeSelector:
    node-role.kubernetes.io/loadbalancer: "" # Load balancer node selector.
  type: L2
```

4. Create a Service resource with the annotation and name MetalLoadBalancerClass:

> **i** When creating a Service resource, you can omit the annotations `network.deckhouse.io/l2-load-balancer-external-ips-count` and `network.deckhouse.io/load-balancer-ips` from the examples below. In this case, the service will automatically be assigned 1 random address from the pool.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  annotations:
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
spec:
  type: LoadBalancer
  loadBalancerClass: ingress # MetalLoadBalancerClass name.
  ports:
  - port: 8000
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
```

If you need to assign specific IP addresses from the pool to the service...

> **i** If the service needs to allocate specific IP addresses from the pool, use the annotation `network.deckhouse.io/load-balancer-ips`. If there is more than one desired address, there must also be an annotation `network.deckhouse.io/l2-load-balancer-external-ips-count`, which must specify the number of addresses allocated from the pool (it must not be less than the number of addresses listed in `network.deckhouse.io/load-balancer-ips`).

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  annotations:
    # The number of addresses that will be allocated from the pool declared in
MetalLoadBalancerClass.
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
    # A list of addresses from the pool declared in MetalLoadBalancerClass that will be allocated to
the service.
    network.deckhouse.io/load-balancer-ips: "192.168.2.102,192.168.2.103,192.168.2.104"
spec:
  type: LoadBalancer
  loadBalancerClass: ingress # MetalLoadBalancerClass name.
  ports:
  - port: 8000
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
```

As a result, the LoadBalancer Service will be assigned the configured number of addresses:

```
d8 k get svc
```

Example output:

```
NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)        AGE
nginx-deployment        LoadBalancer   10.222.130.11   192.168.2.100,192.168.2.101,192.168.2.102
80:30544/TCP   11s
```

The obtained `EXTERNAL-IP` can be set as A records for the application domain:

```
curl -s -o /dev/null -w "%{http_code}" 192.168.2.100:8000
curl -s -o /dev/null -w "%{http_code}" 192.168.2.101:8000
curl -s -o /dev/null -w "%{http_code}" 192.168.2.102:8000
```

Example output:

```
200
```

BGP mode

In BGP mode, `metallb` provides LoadBalancer-type services in Kubernetes clusters deployed on physical infrastructure. Service IP addresses are announced directly to routers (or top-of-rack switches) via the BGP protocol.

How it works

Configuration

- Define an IP address pool available for Service allocation.
- Specify BGP session parameters: the Kubernetes cluster Autonomous System (AS) number, router (peer) IP addresses, peer AS numbers, and passwords for authentication (if needed).
- For each IP pool, define specific announcement parameters, such as community strings.

Establishing BGP sessions

- On each Kubernetes node running `metallb`, the `speaker` component establishes BGP sessions with the configured routers.
- Route information is exchanged between `metallb` and the routers.

Assigning IP addresses to Services

- When a LoadBalancer Service is created, `metallb` selects a free IP address from the configured pool and assigns it to the Service.
- The `controller` component tracks Service changes and manages IP address assignments.

Announcing IP addresses

- After assigning an IP address, the `speaker` component on one of the nodes (leader for the Service) announces it via the established BGP sessions.
- Routers receive the announcement and update their routing tables to direct traffic to the corresponding node.

Traffic distribution

- Routers use Equal-Cost Multi-Path (ECMP) protocols or other load balancing algorithms to distribute traffic between nodes announcing the same Service IP.
- Once delivered to the node, incoming traffic is forwarded to the Service pods using the CNI mechanisms (iptables/IPVS, eBPF, etc.).

Benefits of using BGP

- BGP is supported by most network equipment.
- The network can include multiple routers and many nodes.
- Traffic balancing via ECMP.
- If a node stops announcing an IP, routers automatically redirect traffic to other nodes with the same IP.

Drawbacks of using BGP

- More complex to configure than ARP/GARP announcements (Layer 2 mode).
- Routers must support BGP and ECMP.

Example of using MetalLB in BGP LoadBalancer mode

1. Enable the `metallb` module and configure the required parameters:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  settings:
    addressPools:
    - addresses:
      - 192.168.219.100-192.168.219.200
      name: mypool
      protocol: bgp
    bgpPeers:
    - hold-time: 3s
      my-asn: 64600
      peer-address: 172.18.18.10
      peer-asn: 64601
    speaker:
      nodeSelector:
        node-role.deckhouse.io/metallb: ""
  version: 2
```

2. Configure BGP peering on the network equipment.

Additional resources

- Module `metallb` documentation

Balancing in clusters on cloud platforms

A list of supported providers:

- Amazon Web Services
- Google Cloud Platform
- Microsoft Azure
- OpenStack
- Huawei Cloud
- VMware Cloud DirectorExperimental
- VMware vSphere
- Yandex Cloud
- zVirtExperimental

Configuring incoming traffic balancing in clusters on cloud platforms involves creating an Ingress controller with specified LoadBalancer parameters. Based on these settings, the cloud provider automatically creates an external load balancer. In the cluster, a Service resource is created through which the external load balancer routes traffic to applications.

Example of creating an Ingress controller for the OpenStack provider

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

Example of creating a ClusterIP-type Service

```
apiVersion: v1
kind: Service
metadata:
  name: backend-resolver-cluster-ip
spec:
  ports:
  - name: http
    port: 8000
    protocol: TCP
  selector:
    app: lab-4-backend
  type: ClusterIP
```

Example for VK Cloud

The following example is relevant when the internal balancer would be used.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: nginx
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    annotations:
      service.beta.kubernetes.io/openstack-internal-load-balancer: "true"
  nodeSelector:
    node.deckhouse.io/group: worker
```

ALB with Ingress NGINX Controller

The `ingress-nginx` module is used to implement ALB using the Ingress NGINX Controller.

> **i**
>
> In 2025, Ingress NGINX was placed in maintenance mode, with no plans for active development of new features. Further evolution of inbound traffic load balancing in Kubernetes is focused on the Gateway API.
>
> This does not apply to the module as part of Deckhouse Kubernetes Platform: the module is maintained by the Deckhouse team, including security updates. For details, see Module support and security.

The `ingress-nginx` module installs the Ingress NGINX Controller and manages it with custom resources. If there is more than one node available for hosting the Ingress controller, it is deployed in the HA mode, taking into account the infrastructure specifics of both cloud and bare-metal environments, as well as various Kubernetes cluster types.

The module supports running and configuring several Ingress NGINX controllers simultaneously (one of the controllers is the **primary** one; you can create as many **additional** controllers as you want). This approach allows you to separate extranet and intranet Ingress resources of applications.

Traffic termination options

Traffic to `ingress-nginx` can be routed in several ways:

- Directly without the use of an external load balancer.
- Using an external LoadBalancer; the following variants are supported:
  - Qrator
  - Cloudflare
  - AWS LB
  - GCE LB
  - ACS LB
  - Yandex LB
  - OpenStack LB

HTTPS termination

The module allows you to manage HTTPS security policies for each of the Ingress NGINX controllers, including:

- HSTS parameters
- Available SSL/TLS versions and encryption protocols

The module is integrated with the `cert-manager` module. Thus, it can get SSL certificates automatically and pass them to Ingress NGINX controllers for further use.

Monitoring and statistics

The current `ingress-nginx` implementation has a Prometheus-based system for collecting statistical data. It uses a variety of metrics based on:

- The overall and upstream response time
- Response codes
- Number of repeated requests (retries)
- Request and response sizes
- Request methods
- `content-types`
- Geography of requests, etc.

The data can be grouped by the:

- `namespace`
- `vhost`
- `ingress` resources
- `location` (in nginx)

All graphs are grouped by Grafana dashboards. Also, you can do a drill-down for any graph: for example, from a `namespace` statistics view, you can click through to the corresponding `vhost` dashboard for more detail, and continue down the hierarchy.

Statistics

Basic principles of collecting statistics

1. At the `log_by_lua_block` stage, the module calculates the necessary metrics for each request and stores them in a buffer (each NGINX worker has its own buffer).
2. At the `init_by lua block` stage, each NGINX worker starts a process that sends data in `protobuf` format via TCP socket to the `protobuf_exporter` every second (developed by Deckhouse Kubernetes Platform).
3. `protobuf_exporter` runs as a sidecar container in the Ingress controller pod, receives `protobuf` messages, parses and aggregates them, and exports metrics for Prometheus.
4. Prometheus scrapes metrics every 30 seconds from both the Ingress controller and the `protobuf_exporter`. This scraped data is what statistics is based on.

Metrics structure and representation

All collected metrics include service labels identifying the controller instance: `controller`, `app`, `instance`, and `endpoint` (visible in `/prometheus/targets`).

- All non-geo metrics exported by `protobuf_exporter` are provided at three detail levels:
  - `ingress nginx overall *`: Top-level aggregated metrics (non-detailed, all metrics have the following labels: `namespace`, `vhost`, `content_kind`).
  - `ingress nginx_detail_*`: In addition to overall metrics, adds `ingress`, `service`, `service_port`, and `location`.
  - `ingress_nginx_detail_backend_*`: Backend-level metrics. In addition to detail metrics, adds the `pod_ip` label.
- Metrics collected for overall and detail levels:
  - `* requests total`: Total requests (extra labels: `scheme`, `method`).
  - `* responses total`: Number of responses (extra label: `status`).
  - `* request seconds {sum,count,bucket}`: Response time histogram.
  - `* bytes received {sum,count,bucket}`: Request size histogram.
  - `* bytes sent {sum,count,bucket}`: Response size histogram.
  - `*_upstream_response_seconds_{sum,count,bucket}`: Upstream service response time histogram (total for multiple upstreams).
  - `*_lowres_upstream_response_seconds_{sum,count,bucket}`: Simplified histogram (for visualization; can't be used for quantiles).
  - `*_upstream_retries_{count,sum}`: Number and total of backend retries.
- Metrics collected for overall level:
  - `*_geohash_total`: Request counts per geohash (additional labels: `geohash`, `place`).
- Metrics collected for detail_backend level:
  - `* lowres upstream_response_seconds`: Simplified response time histogram for overall and detail.
  - `* responses total`: Number of responses (additional label: `status_class`, not just `status`).
  - `*_upstream_bytes_received_sum`: Total size of data received from backends.

Load balancing configuration examples

Use the IngressNginxController custom resource to configure load balancing.

Example for AWS (Network Load Balancer)

When setting up the balancer, all available zones in the cluster are used.

Each zone's balancer receives its own public IP. If a zone has an Ingress controller instance, its IP is added to the load balancer's domain name as an A record.

If no instances remain in a zone, that IP is removed from DNS.

If only one Ingress controller instance exists in a zone, its IP is temporarily removed from DNS during pod restarts.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
 name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
```

Example for GCP / Yandex Cloud / Azure

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
 name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
```

i In GCP, nodes must have an annotation allowing external connections for NodePort services.

Example for OpenStack

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main-lbwpp
spec:
  inlet: LoadBalancerWithProxyProtocol
  ingressClass: nginx
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
```

Example for VK Cloud

The following example is relevant when the internal balancer would be used.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: nginx
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    annotations:
      service.beta.kubernetes.io/openstack-internal-load-balancer: "true"
  nodeSelector:
    node.deckhouse.io/group: worker
```

Example for bare metal

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: HostWithFailover
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    value: frontend
```

Example for bare metal with external load balancer

The following example is relevant when using Cloudflare, Qrator, Nginx+, Citrix ADC, Kemp or other external load balancers.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: HostPort
  hostPort:
    httpPort: 80
    httpsPort: 443
    behindL7Proxy: true
```

Example for bare metal (MetalLB in BGP LoadBalancer mode)

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    value: frontend
```

When using MetalLB, its speaker pods must run on the same nodes as the Ingress controller pods.

To preserve the real client IP addresses, the Ingress controller Service should be created with `externalTrafficPolicy: Local` to avoid inter-node SNAT. In this configuration, MetalLB speaker will only announce the Service from nodes running target pods.

Example `metallb` configuration:

```
metallb:
 speaker:
   nodeSelector:
     node-role.deckhouse.io/frontend: ""
   tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend
```

Example for bare metal (MetalLB in L2 LoadBalancer mode)

1. Enable the `metallb` module:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  version: 2
```

2. Create a MetalLoadBalancerClass resource:

> **i** Metallb balancers should be placed on the same nodes as ingress controllers. In typical deployment scenarios, frontend nodes are used for this purpose (to deploy ingress controllers and Metallb load balancers on frontend nodes, use the annotation `node-role.deckhouse.io/frontend: ""` in their manifests).

```
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: ingress
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  isDefault: false
  nodeSelector:
    node-role.deckhouse.io/frontend: "" # Load balancer node selector.
  type: L2
```

3. Create a IngressNginxController resource:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    loadBalancerClass: ingress
    annotations:
      # Number of addresses to allocate from the pool defined in MetalLoadBalancerClass.
      network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    value: frontend
    operator: Equal
```

> **i** When creating an ingress controller, you can also specify certain IP addresses from the pool that will be assigned to it. To specify the addresses that should be assigned to the service, use the annotation `network.deckhouse.io/load-balancer-ips`. If there is more than one desired address, there must also be an annotation `network.deckhouse.io/l2-load-balancer-external-ips-count`, which must specify the number of addresses allocated from the pool (it must not be less than the number of addresses listed in `network.deckhouse.io/load-balancer-ips`). Example of using annotations to assign specific addresses from the pool to the service.

The platform will create a LoadBalancer Service with the specified number of IPs:

```
d8 k -n d8-ingress-nginx get svc
```

Example output:

```
NAME                  TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)                      AGE
main-load-balancer    LoadBalancer   10.222.130.11   192.168.2.100,192.168.2.101,192.168.2.102
80:30689/TCP,443:30668/TCP   11s
```

Example of segregating access between public and administrative zones

In many applications, the same backend serves both the public part and the administrative interface. For example:

- `https://example.com` is the public zone;
- `https://admin.example.com` is the administrative zone, access to which must be restricted (`ACL`, `mTLS`, `IP whitelist`, and so on).

For this scenario, we recommend offloading administrative traffic to a separate Ingress controller (with a dedicated Ingress class if necessary) and restricting access to it by using the `spec.acceptRequestsFrom` parameter.

Specifics of using a single Ingress controller

Consider an example where a single Ingress controller is used to serve requests from both the public zone and the administrative interface.

Example of Ingress resource configuration for this case:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: admin-ingress
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: "1.2.3.4/32"
spec:
  ingressClassName: nginx # The Ingress resource for administrative traffic is associated with the
same Ingress controller as the Ingress resource for public traffic.
  rules:
    - host: admin.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: public-ingress
spec:
  ingressClassName: nginx # The Ingress resource for public traffic is associated with the same
Ingress controller as the Ingress resource for administrative traffic.
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
```

With processing and forwarding of X-Forwarded-* headers enabled, the backend can rely on the `x-forwarded-host` header when making authorization decisions. In the example above, it is possible to reach the administrative zone through the Ingress resource that serves public traffic by using `x-forwarded-host`. Therefore, when using this option you must be sure that requests to the Ingress controller come only from trusted sources.

Using separate Ingress controllers

To avoid the situation described above (when, with processing and forwarding of X-Forwarded-* headers enabled it is possible, for example, to reach the administrative zone via the Ingress resource that serves public traffic by using `x-forwarded-host`), we recommend that you:

- configure access rules at the Ingress resource level,
- use separate Ingress controllers,
- restrict which source addresses are allowed to connect to the Ingress controllers.

Example of Ingress resource configuration for this case:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: admin-ingress
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: "1.2.3.4/32"
spec:
  ingressClassName: admin-nginx # The Ingress resource for administrative traffic is associated with
a separate Ingress controller.
  rules:
    - host: admin.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: public-ingress
spec:
  ingressClassName: public-nginx # The Ingress resource for public traffic is associated with a
separate Ingress controller.
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
```

Example of an Ingress controller that serves administrative Ingress resources and accepts connections only from specified subnets:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: admin
spec:
  ingressClass: admin-nginx
  inlet: HostPort
  acceptRequestsFrom:
    - 1.2.3.4/32
    - 10.0.0.0/16
  hostPort:
    httpPort: 80
    httpsPort: 443
    behindL7Proxy: true
```

In this example:

- The Ingress controller is exposed on node ports through the `HostPort` inlet.
- The `acceptRequestsFrom` parameter allows connections to the controller only from the listed subnets.
- Even if an external load balancer or client can set its own `X-Forwarded-*` header values, the decision whether to allow the connection to reach the controller is made based on the actual source address, not on headers.
- Administrative Ingress resources (in this example `admin-ingress`) are served by this controller according to the configured Ingress class.

Example of an Ingress controller that serves Ingress resources for public traffic:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: public
spec:
  ingressClass: public-nginx
  inlet: HostPort
  hostPort:
    httpPort: 8080
    httpsPort: 8443
    behindL7Proxy: true
```

Module support and security

The `ingress-nginx` module is covered by Deckhouse Kubernetes Platform maintenance for the entire platform support lifecycle, regardless of the upstream project’s development status. The Deckhouse team tracks CVEs in the controller and its dependencies — NGINX, Lua modules, and base images — and delivers fixes in platform releases. For compliance with PCI DSS expectations regarding vendor support and vulnerability remediation timelines, Flant is the responsible vendor of the module. Among other things, vulnerability management processes and the release of security updates.

Additional resources

- Module `cert-manager` documentation
- Module `ingress-nginx` documentation
- Module `metallb` documentation

ALB with Kubernetes Gateway API

To implement ALB using the Kubernetes Gateway API, the `alb` module is used.

The `alb` module implements an Application Load Balancer (ALB) and allows you to publish applications through Kubernetes Gateway API. It deploys and configures the infrastructure for receiving and routing external requests, and also verifies the user configuration of the Gateway API.

> **i** ALBs created using the Kubernetes Gateway API can be used in a cluster alongside ALBs created using the NGINX Controller Ingress. For more information, see the section Using with other modules and third-party solutions.

Validating Gateway API configuration

In addition to Gateway API infrastructure configuration, the `alb` module validates user settings to prevent conflicting configurations from being applied. For example, the module checks for conflicts between identical traffic handlers in different ListenerSet objects when they point to the same Gateway object.

Steps to take before enabling and configuring ALB in a cluster

Before enabling and configuring ALB in a DKP cluster:

- If you need to publish service domains (web interfaces for DKP service components and other modules), ensure that the global parameter `publicDomainTemplate` is specified. If the `publicDomainTemplate` parameter is not specified, HTTPRoute/Gateway/ListenerSet system objects will be created incorrectly, and the web interfaces of DKP service components and other modules will not be published.
- If third-party Gateway API solutions are used in the cluster, check compatibility between the API versions used for third-party Gateway API objects and the versions required by the `alb` module controller.

Using with other modules and third-party solutions

ALBs implemented using the Kubernetes Gateway API in a DKP cluster can be used in conjunction with ALBs implemented using the NGINX Controller Ingress, as well as with ALBs based on third-party Gateway API solutions.

> **i** For the DKP gateway, a ClusterIssuer object is automatically created by default. This same ClusterIssuer object is used to issue certificates for system domains.

Using third-party Gateway API solutions

Use of third-party Gateway API implementations is supported, provided that the cluster uses the following Gateway API object storage versions compatible with the `alb` module controller:

- BackendTLSPolicy: v1;
- GatewayClass: v1;
- Gateway: v1;
- ListenerSet: v1;
- GRPCRoute: v1;
- HTTPRoute: v1;
- ReferenceGrant: v1beta1;
- TCPRoute: v1alpha2;
- TLSRoute: v1.

During startup, the `alb` module controller checks the currently stored versions of Gateway API objects. If it detects a mismatch between the installed and required versions, it stops and does not proceed. If a given Gateway API object type is completely absent from the cluster, the controller automatically creates the required CRD version and then continues startup.

To manually verify version compatibility of the installed Gateway API objects in the cluster, use:

```
declare -A want=(
    [gatewayclasses.gateway.networking.k8s.io]=v1
    [gateways.gateway.networking.k8s.io]=v1
    [grpcroutes.gateway.networking.k8s.io]=v1
    [httproutes.gateway.networking.k8s.io]=v1
    [listenersets.gateway.networking.k8s.io]=v1
    [referencegrants.gateway.networking.k8s.io]=v1beta1
    [tcproutes.gateway.networking.k8s.io]=v1alpha2
    [tlsroutes.gateway.networking.k8s.io]=v1
    [backendtlspolicies.gateway.networking.k8s.io]=v1
)

for crd in "${!want[@]}"; do
    got="$(
       d8 k get crd "$crd" -o jsonpath='{.spec.versions[?(@.storage==true)].name}'
2>/dev/null || true
    )"
    if [[ "$got" == "${want[$crd]}" ]]; then
         echo "$crd OK storage=$got"
    else
         echo "$crd FAILED cluster=${got:-MISSING} expected=${want[$crd]}"
    fi
done | sort
```

Otherwise, the module only configures and manages Gateway objects associated with its designated GatewayClass, which minimizes the risk of conflicts when third-party Gateway API implementations are present.

Publishing an application

The process of publishing an application includes the following steps:

1. Creating a Gateway object using the ClusterALBInstance resource (cluster-scoped) or the ALBInstance resource (namespaced).
2. Create a ListenerSet object (manages the reception of incoming requests), which is bound to the Gateway object created in the previous step.
3. Create objects (routes) to route incoming requests to the application and bind them to the ListenerSet. The HTTPRoute, GRPCRoute, TCPRoute, and TLSRoute objects are used for routing (the appropriate one is selected based on the type of traffic to the published application).

Creating a Gateway object

When creating a Gateway managed object for publishing user applications, the custom resources ClusterALBInstance (a cluster-scoped object) and ALBInstance (a namespaced resource) are used.

The characteristics of these resources and the differences between them are described in the table:

	ClusterALBInstance	**ALBInstance**
Purpose	Deploy a cluster-wide Gateway object	Deploy a local Gateway object
Typical use case	- Common entry point (cluster-wide gateway). - System gateway for publishing web interfaces of DKP service components and other modules (may require cluster preparation). - Platform gateway	Dedicated gateway for an application or team in a dedicated namespace
Supported inlet types	`LoadBalancer`, `HostPort`	`LoadBalancer`
Proxy implementation	Envoy Proxy	Envoy Proxy
Deployment type	DaemonSet	Deployment
Placement of ListenerSet objects and routes	In any user namespace	In the same namespace as the ALBInstance object

	ClusterALBInstance	ALBInstance
Access level	Cluster administrator	Namespace administrator

Creating a ClusterALBInstance object or an ALBInstance object results in creation of a managed Gateway object in the cluster. At the same time:

- Each Gateway object is served by at least one Envoy Proxy instance.
- Traffic reaches it through a Service object of type `LoadBalancer` or directly by using `HostPort` parameters.
- Each Gateway object creates two default listeners: `d8-http` (port `80`) and `d8-https` (port `443`). They are intended for service tasks such as gateway availability checks or cert-manager HTTP-01 challenges. They are not recommended for publishing applications; use ListenerSet for that purpose instead.

⚠ Manual modification of Gateway objects managed by the module is not allowed.

Example of a ClusterALBInstance resource manifest for creating a cluster-wide gateway:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterALBInstance
metadata:
  name: public-gw
  namespace: prod
spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
```

An example of an ALBInstance resource manifest for creating a separate ingress for an application or a team in a dedicated namespace is provided in the Usage section.

Creating ListenerSet objects to manage the handling of incoming requests

The ListenerSet object describes system and user traffic handlers that define hostname, TLS mode, port, and protocol. Each ListenerSet object is linked to a specific parent Gateway object through the `spec.parentRef` field, and routes are then attached to it.

Placement of ListenerSet objects depends on the type of Gateway object in use:

- for ClusterALBInstance, ListenerSet objects may be placed in any namespace;
- for ALBInstance, ListenerSet objects are recommended to be placed in the same namespace.

In both cases, it is recommended to place the ListenerSet object in the same namespace as the HTTPRoute, GRPCRoute, TCPRoute, and TLSRoute objects attached to it. This improves configuration readability and helps avoid additional setup such as ReferenceGrant objects.

An example of a ListenerSet resource manifest for managing the reception of incoming HTTP and HTTPS requests through a cluster-wide gateway:

```
apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: app-listeners
  namespace: prod
spec:
  parentRef:
    name: public-gw  # The name of the Gateway object from the ClusterALBInstance
status, provided by the administrator.
    namespace: d8-alb
  listeners:
    - name: app-http
      port: 80 # HTTP traffic always uses 80 regardless of ClusterALBInstance
settings.
      protocol: HTTP
      hostname: app.example.com
    - name: app-https
      port: 443 # HTTPS traffic always uses 443 regardless of ClusterALBInstance
settings.
      protocol: HTTPS
      hostname: app.example.com
      tls:
        mode: Terminate
        certificateRefs:
          - name: app-tls  # Reference to the secret with the TLS certificate.
            namespace: prod
```

Creating routes and configuring routing

The following route types are used to route incoming requests:

- HTTPRoute: For routing HTTP/HTTPS/TLS requests. HTTPRoute objects support extended settings through annotations that complement the current Gateway API specification.
- GRPCRoute: For routing gRPC traffic.
- TLSRoute: For TLS passthrough routing.
- TCPRoute: For routing TCP traffic.

Example of a route for HTTP traffic:

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: app-http-route
  namespace: prod
spec:
  parentRefs:
    - name: app-listeners # ListenerSet name.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: app-http
      port: 80
  hostnames:
    - app.example.com
  rules:
    - backendRefs:
        - name: app-svc # Reference to the internal load balancer of the application.
          port: 8080
```

Example of a route for HTTPS traffic:

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: app-https-route
  namespace: prod
spec:
  parentRefs:
    - name: app-listeners # ListenerSet name.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: app-https
      port: 443 # HTTPS traffic always uses 443 regardless of ClusterALBInstance
settings.
  hostnames:
    - app.example.com
  rules:
    - backendRefs:
        - name: app-svc # Reference to the internal load balancer of the application.
          port: 8080
```

Publishing service domains

⚠ If you need to publish the service domains, ensure that the global parameter `publicDomainTemplate` is specified. If it is not specified, system HTTPRoute/Gateway/ListenerSet objects will be created incorrectly, and the web interfaces of DKP service components and other modules will not be published.

To provide access to the DKP cluster's service domains, specify a default gateway. To do this, follow these steps:

1. Create a cluster-scoped ClusterALBInstance object with the desired inlet type and configuration. Set the `spec.defaultDeckhouseGateway: true` parameter for this ClusterALBInstance.

   Example of a manifest for a cluster-scoped ClusterALBInstance object with a parameter `spec.defaultDeckhouseGateway: true`:

   ```
   kind: ClusterALBInstance
   metadata:
     name: public-gw
   spec:
     gatewayName: public-gw
     defaultDeckhouseGateway: true
     inlet:
       type: LoadBalancer
   ```

2. After applying the changes, check the status of the ClusterALBInstance object:

   ```
   d8 k get clusteralbinstances
   ```

   The ClusterALBInstance object must expose the managed Gateway object, and the instance itself must move to a ready state. After that, system ListenerSet objects and HTTPRoute objects must appear in the corresponding system namespaces of the cluster.

Algorithm for selecting the default DKP gateway when using multiple ClusterALBInstances

A cluster can have multiple cluster-scoped gateways simultaneously, each marked as the default gateway (with the `spec.defaultDeckhouseGateway: true` flag set for the corresponding ClusterALBInstance). In that case, the default gateway is the Gateway object created by the oldest ClusterALBInstance object according to `creationTimestamp`. If no ClusterALBInstance object is marked as the default gateway, DKP allows the Gateway object created by the `alb` module for the instance

named `main` to be used as the default gateway.

Changing the default DKP gateway

If DKP system domains need to move to another Gateway object, complete these steps:

1. Create a new ClusterALBInstance object that describes the required settings and set `spec.defaultDeckhouseGateway: true` on it.
2. In the current ClusterALBInstance object that provides the default DKP gateway, set `spec.defaultDeckhouseGateway: false`.
3. Check that all system ListenerSet objects now point to the new Gateway object in `spec.parentRef`.

Changing the inlet while keeping the current Gateway

To change the inlet used for an existing Gateway object, complete these steps:

1. Create a new ClusterALBInstance object or ALBInstance object with a different name but the same `spec.gatewayName`, using the required inlet type.
2. Check that the new traffic path works correctly.
3. Delete the obsolete ClusterALBInstance object or ALBInstance object.

Because `gatewayName` does not change, the Gateway object stays the same. In most cases, the ListenerSet object and routes do not need to be rewritten.

Opening an additional TCP port on the cluster-wide gateway

If a dedicated TCP port is needed in addition to the default HTTP/HTTPS listeners, add the `spec.inlet.additionalPorts` field to the corresponding ClusterALBInstance object, for example:

```
...

spec:
  gatewayName: public-gw
  inlet:
    type: LoadBalancer
    loadBalancer: {}
    additionalPorts:
      - port: 9000
        protocol: TCP

...
```

The controller adds a corresponding TCP traffic handler to the managed Gateway object with a section name (`sectionName`) like `tcp-port-9000`. Then a TCPRoute object may be created that points directly to that Gateway object and that `sectionName`:

```
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: TCPRoute
metadata:
  name: app-tcp
  namespace: prod
spec:
  parentRefs:
    - name: public-gw
      namespace: d8-alb
      sectionName: tcp-port-9000
      port: 9000
  rules:
    - backendRefs:
        - name: tcp-svc
          port: 9000
```

> **i** If a TCPRoute object is created in a namespace different from the Gateway object namespace, a corresponding ReferenceGrant object must be created.

If the same Gateway object is shared by several ClusterALBInstance objects, the `additionalPorts` set that actually reaches the Gateway object comes from the oldest ClusterALBInstance object. The others may report port conflicts in status.

Viewing Envoy Proxy Configuration

For troubleshooting, it is useful to inspect the configuration that the controller and the proxy configurator pushed into the Envoy Proxy instance that serves the Gateway object.

To do this, follow these steps:

1. Select an Envoy Proxy pod for the required Gateway object:

   ```
   d8 k -n d8-alb get pods -l alb.deckhouse.io/gateway=shared-gateway
   ```

2. Get the configuration through the following command:

   ```
   d8 k -n d8-alb exec -it <envoy-proxy-pod-name> pilot-agent request GET /config_dump
   ```

   If only one section of the configuration is needed, the required section may be requested explicitly:

   ```
   d8 k -n d8-alb exec -it <envoy-proxy-pod-name> pilot-agent request GET /config_dump?
   resource=dynamic_listeners
   d8 k -n d8-alb exec -it <envoy-proxy-pod-name> pilot-agent request GET /config_dump?
   resource=dynamic_route_configs
   d8 k -n d8-alb exec -it <envoy-proxy-pod-name> pilot-agent request GET /config_dump?
   resource=dynamic_active_clusters
   ```

This makes it easy to check whether the expected traffic handlers, virtual hosts, and upstream clusters appeared after changes to the ListenerSet object or Route object.

Additional resources

- Global parameters
- Module `alb` documentation

ALB with Istio

ALB with Istio is implemented via Istio Ingress Gateway or Istio Ingress Gateway or Ingress NGINX. The `istio` module is used for this purpose.

Ingress to publish applications

Istio Ingress Gateway

To publish the application using the Istio Ingress Gateway, follow these steps:

1. Create an IngressIstioController resource:

```
apiVersion: deckhouse.io/v1alpha1
kind: IngressIstioController
metadata:
 name: main
spec:
  # ingressGatewayClass contains the label selector value used to create the
Gateway resource.
  ingressGatewayClass: istio-hp
  inlet: HostPort
  hostPort:
    httpPort: 80
    httpsPort: 443
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
  resourcesRequests:
    mode: VPA
```

2. Create a secret that defines the TLS certificate and key for securing HTTPS traffic:

```
apiVersion: v1
kind: Secret
metadata:
  name: app-tls-secret
  namespace: d8-ingress-istio # Note that the namespace is not app-ns.
type: kubernetes.io/tls
data:
  tls.crt: |
    <tls.crt data>
  tls.key: |
    <tls.key data>
```

3. Create a Gateway resource. In the `spec.selector` field, specify the label that references the ingressGatewayClass and the secret created in the previous steps:

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: gateway-app
  namespace: app-ns
spec:
  selector:
    # Label selector for using the Istio Ingress Gateway main-hp.
    istio.deckhouse.io/ingress-gateway-class: istio-hp
  servers:
    - port:
       # Standard template for using the HTTP protocol.
       number: 80
       name: http
       protocol: HTTP
      hosts:
        - app.example.com
    - port:
       # Standard template for using the HTTPS protocol.
       number: 443
       name: https
       protocol: HTTPS
      tls:
        mode: SIMPLE
        # A Secret resource with a certificate and a key, which must be created
in the d8-ingress-istio namespace.
        credentialName: app-tls-secret
      hosts:
        - app.example.com
```

Supported Secret formats can be found at the official website of the Istio project.

4. Define routing rules using a VirtualService that links the gateway to the service it serves:

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: vs-app
  namespace: app-ns
spec:
  gateways:
    - gateway-app # Gateway name
  hosts:
    - app.example.com
  http:
    - route:
       - destination:
          host: app-svc # Service name
```

Ingress NGINX

To use Ingress NGINX, you need to:

- Configure the Ingress controller by adding Istio sidecar to it. Enable the `enableIstioSidecar` parameter in the `ingress-nginx` module's IngressNginxController custom resource.
- Set up an Ingress that refers to the Service. The following annotations are mandatory for Ingress:
  - `nginx.ingress.kubernetes.io/service-upstream: "true"`: Using this annotation, the Ingress controller sends requests to the Service ClusterIP (from Service CIDR) instead of sending them directly to the application's pods. This is required because the `istio-proxy` sidecar container only catches traffic directed to Service CIDR. Any requests out of this range do not go through Istio.
  - `nginx.ingress.kubernetes.io/upstream-vhost: productpage.bookinfo.svc`: Using this annotation, the sidecar container can identify the application service that serves requests.

Examples:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: productpage
  namespace: bookinfo
  annotations:
    # Nginx proxies traffic to the ClusterIP instead of pods' own IPs.
    nginx.ingress.kubernetes.io/service-upstream: "true"
    # In Istio, all routing is carried out based on the `Host:` headers.
    # Instead of letting Istio know about the `productpage.example.com` external
domain,
    # we use the internal domain of which Istio is aware.
    nginx.ingress.kubernetes.io/upstream-vhost: productpage.bookinfo.svc
spec:
  rules:
    - host: productpage.example.com
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: productpage
              port:
                number: 9080
```

```
apiVersion: v1
kind: Service
metadata:
  name: productpage
  namespace: bookinfo
spec:
  ports:
  - name: http
    port: 9080
  selector:
    app: productpage
  type: ClusterIP
```

Additional resources

- Module `ingress-nginx` documentation
- Module `istio` documentation

Ensuring high availability and fault tolerance (keepalived)

In Deckhouse Kubernetes Platform, the `keepalived` module can be used to provide high availability and fault tolerance.

To configure keepalived clusters, custom resources are used.

Module usage examples

Multiple public IP addresses

In the following example, there are three public IP addresses on three front nodes. Each virtual IP address is assigned to a separate VRRP group. Thus, each address “jumps” independently of the others, and if there are three nodes in the cluster with the `node-role.deckhouse.io/frontend: ""` labels, each IP address gets its own master node.

```
apiVersion: deckhouse.io/v1alpha1
kind: KeepalivedInstance
metadata:
  name: front
spec:
  nodeSelector: # Required.
    node-role.deckhouse.io/frontend: ""
  tolerations:  # Optional.
  - key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
  vrrpInstances:
  - id: 1 # The unique cluster ID.
    interface:
      detectionStrategy: DefaultRoute # The card with the default route is used as a
service network one.
    virtualIPAddresses:
    - address: 42.43.44.101/32
      # The interface parameter is omitted since IP addresses are based on the cards
that service VRRP traffic.
  - id: 2
    interface:
      detectionStrategy: DefaultRoute
    virtualIPAddresses:
    - address: 42.43.44.102/32
  - id: 3
    interface:
      detectionStrategy: DefaultRoute
    virtualIPAddresses:
    - address: 42.43.44.103/32
```

In the following example, there is a gateway with a pair of IP addresses for LAN and WAN. In the case of the gateway, the private and public IPs are bind together, and they will “jump” between the nodes in tandem. In the below example, the VRRP traffic is routed through the LAN interface. It can be detected using the NetworkAddress method (assuming that each node has an IP address belonging to this subnet).

```
apiVersion: deckhouse.io/v1alpha1
kind: KeepalivedInstance
metadata:
  name: mygateway
spec:
  nodeSelector:
    node-role.deckhouse.io/mygateway: ""
  tolerations:
  - key: node-role.deckhouse.io/mygateway
    operator: Exists
  vrrpInstances:
  - id: 4 # Since "1", "2", "3" IDs are used in the "front" KeepalivedInstance above.
    interface:
      detectionStrategy: NetworkAddress
      networkAddress: 192.168.42.0/24
    virtualIPAddresses:
    - address: 192.168.42.1/24
      # In this case, we have already detected the local network (above); thus, the
interface parameter can be safely omitted.
    - address: 42.43.44.1/28
      interface:
        detectionStrategy: Name
        name: ens7 # The interface for public IPs is called "ens7" on all nodes,
therefore it needs to be named explicitly.
```

Switching keepalived manually

1. Enter the desired pod using a debug container with a shared process namespace:

   ```
   d8 k debug -n d8-keepalived -it keepalived-<name> --profile=general --target keepalived
   ```

2. Edit the configuration file `vim /proc/1/root/etc/keepalived/keepalived.conf`, replace the value in the `priority` line with <number of keepalived pods + 1> or set a value higher than the current VRRP master (e.g., `255`).

3. Apply settings – send a signal to reload the configuration:

   ```
   kill -HUP 1
   ```

Additional resources

- Module `keepalived` documentation

Egress Gateway configuration

Egress Gateway enables centralized management and control of outgoing traffic and provides features such as encryption, routing, and monitoring.

⚠ To use the Egress Gateway, enable the `cni-cilium` module in the cluster.

Egress Gateway usage modes

Basic mode

Pre-configured IP addresses are used on egress nodes.

Virtual IP mode

Egress Gateway lets you dynamically assign additional IP addresses to nodes.

Configuration principle

Configuring an egress gateway requires two custom resources:

1. EgressGateway: Describes the group of nodes, one of which will be selected as the active egress gateway, while the rest will remain in standby mode:
   - Among the group of nodes matching the `spec.nodeSelector` the eligible nodes will be selected. One of them will be assigned as the active gateway. The active node is selected in alphabetical order.

     Attributes of an eligible node:

     - The node is in `Ready` state.
       - The node is not in the maintenance state (not cordoned).
       - The `cilium-agent` on the node is in the `Ready` state.
     - When using EgressGateway in `VirtualIP` mode, an agent is launched on the active node which emulates a “virtual” IP address using the ARP protocol. The status of this agent’s pod is also taken into account when determining the eligibility of a node.
     - Different EgressGateways can use the same nodes for operation. The active node is selected independently for each EgressGateway, which allows for load balancing between them.
2. EgressGatewayPolicy: Describes the policy for routing network requests from pods in the cluster to a specific egress gateway defined using EgressGateway.

Node maintenance

To perform maintenance on a node that is currently the active egress gateway, follow these steps:

1. Remove the node label to exclude it from the egress gateway candidate pool. `<egress-label>` is the label specified in `spec.nodeSelector` of your EgressGateway.

```
d8 k label node <node-name> <egress-label>-
```

2. Cordon the node to prevent new pods from starting:

   ```
   d8 k cordon <node-name>
   ```

   After this, Cilium will automatically select a new active node from the remaining candidates. Traffic will continue routing through the new gateway without interruption.

3. After maintenance is complete, return the node to service:

   ```
   d8 k uncordon <node-name>
   d8 k label node <node-name> <egress-label>=<value>
   ```

⚠ Reapplying the label may cause the node to become active egress gateway again (if it is first in alphabetical order among candidates).

To avoid immediate return to the active state, temporarily reduce the number of EgressGateway replicas or adjust selection priorities using additional labels.

Comparison with CiliumEgressGatewayPolicy

The `CiliumEgressGatewayPolicy` implies configuring only one node as an egress gateway. If it fails, there are no failover mechanisms and the network connection will be broken.

Configuration examples

EgressGateway in PrimaryIPFromEgressGatewayNodeInterface mode (basic mode)

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGateway
metadata:
  name: my-egressgw
spec:
  nodeSelector:
    node-role.deckhouse.io/egress: ""
  sourceIP:
    mode: PrimaryIPFromEgressGatewayNodeInterface
    primaryIPFromEgressGatewayNodeInterface:
      # The "public" interface must have the same name on all nodes that matching the
nodeSelector.
      # If the active node fails, traffic will be redirected through the backup node
and
      # the source IP address of the network packets will change.
      interfaceName: eth1
```

EgressGateway in VirtualIPAddress mode

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGateway
metadata:
  name: my-egressgw
spec:
  nodeSelector:
    node-role.deckhouse.io/egress: ""
  sourceIP:
    mode: VirtualIPAddress
    virtualIPAddress:
      # Each node must have all the necessary routes configured to access all
external public services,
      # the "public" interface must be prepared for automatic configuration of the
"virtual" IP as a secondary IP address.
      # In case of failure of the active node, traffic will be redirected through the
backup node and
      # the source IP address of the network packets will not change.
      ip: 172.18.18.242
      # List of network interfaces for Virtual IP.
      interfaces:
      - eth1
```

EgressGatewayPolicy

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGatewayPolicy
metadata:
  name: my-egressgw-policy
spec:
  destinationCIDRs:
  - 0.0.0.0/0
  egressGatewayName: my-egressgw
  selectors:
  - podSelector:
      matchLabels:
        app: backend
        io.kubernetes.pod.namespace: my-ns
```

Additional resources

- Module `cni-cilium` parameters
- Module `cni-cilium` custom resources

Inter-cluster cooperation (cluster alliance)

Deckhouse Kubernetes Platform (DKP) supports two inter-cluster communication models:

- Multicluster
- Federation

Both models can be implemented using Istio (via the `istio` module) or Cilium (via the `cni-cilium` module). Both tools provide deployment of a service mesh for managing and configuring network interactions between applications within a cluster.

Service mesh usage specifics in DKP

Available scenarios depend on the platform edition.

Multicluster and federation are available in the Enterprise Edition (EE).

In the Community Edition (CE), Basic Edition (BE), Standard Edition (SE), and Standard Edition+ (SE+), the service mesh can only be used within a single cluster — for example, to implement patterns such as Circuit Breaker, Canary Deployment, and others.

Use case	CE / BE / SE / SE+	EE
Service mesh in one cluster	+	+
Federation	-	+
Multicluster	-	+

Differences between federation and multicluster

Federation connects independent (sovereign) clusters:

- Each cluster has its own namespace and services.
- Inter-cluster communication is configured explicitly. You define which services are accessible externally.

Multicluster tightly connects integrated clusters:

- A shared namespace is used.
- Services from one cluster are available to others as if they were local (unless restricted by authorization policies).

Models for combining multiple clusters (Istio example)

When using Istio, different combination models are possible:

- Network models: Clusters can be in the same or different networks.
- Control plane models: One or multiple service mesh control planes.
- Isolation models: Tenant isolation at the namespace, network, or cluster level.
- Service mesh models: One shared service mesh or multiple meshes.

These parameters can be combined to build an architecture tailored to the requirements of a specific infrastructure.

Problems solved with multicluster and federation

Combining clusters into a federation or multicluster can help solve the following tasks:

- Geographical distribution and GEO load balancing: Clusters serve users closer to their location.
- Fault tolerance: If a cluster or data center fails, traffic automatically switches to others (automatic failover).
- Horizontal scaling: Load distribution across clusters increases overall system performance.
- Environment and resource isolation: Different environments (production, staging, dev) can be placed in separate clusters, with optional interconnection.
- Regulatory compliance: For example, data storage requirements for specific geographic zones.
- Hybrid and multi-cloud infrastructure creation: Combining clusters located in different environments (cloud, different providers, etc.).

Additional resources

- Module `cni-cilium` documentation
- Module `istio` documentation

Multicluster

Multicluster of Istio funds

Requirements for clusters

- Cluster domains in the `clusterDomain` parameter of the ClusterConfiguration resource must be the same for all multicluster members. The default value is `cluster.local`.
- Pod and Service subnets in the `podSubnetCIDR` and `serviceSubnetCIDR` parameters of the ClusterConfiguration resource must be unique for each multicluster member.

ℹ
- When analyzing HTTP and HTTPS traffic *(in Istio terminology)*, you can identify them and decide on further routing or blocking based on their headers.
- At the same time, when analyzing TCP traffic *(in Istio terminology)*, it is possible to identify them and decide on further routing or blocking based only on their destination IP address or port number.

If the IP addresses of services or pods in different clusters match, requests from other pods in other clusters may mistakenly fall under the Istio's rules. The intersection of subnets of services and pods is not recommended (source). Istio operates in the multi-network mode — pods from different clusters can only communicate with each other through the Istio ingress gateway. Direct communication between pods of different clusters is not supported.

General principles

- Multicluster requires mutual trust between clusters to work. This is achieved by a mutual exchange of root certificates between clusters.
- Istio connects directly to the API server of the neighboring cluster to gather information about its services. The Deckhouse `istio` module establishes the secure communication channel.

Enabling the multicluster

Enabling the multicluster (via the `istio.multicluster.enabled = true` module parameter) results in the following:

- A proxy is added to the cluster to publish access to the API server via the standard Ingress resource:
  - Access through this public address is secured by authorization based on Bearer tokens signed with trusted keys.
  - Deckhouse automatically exchanges trusted public keys during the mutual configuration of the multicluster.
  - The proxy has read-only access to a limited set of resources.
- A service gets added to the cluster. That service exports the following cluster metadata to the outside:
  - Istio root certificate (accessible without authentication).
  - The public API server address (available only for authenticated requests from neighboring clusters).
  - List of public addresses of the `ingressgateway` service (available only for authenticated requests from neighboring clusters).
  - Server public keys to authenticate requests to API server and to private metadata.

Managing the multicluster

To create a multicluster, you need to create a set of IstioMulticluster resources in each cluster that describe all the other clusters.

Example of configuring a multicluster of two clusters

Use the IstioMulticluster resource to configure a multicluster using Istio tools.

Cluster A:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioMulticluster
metadata:
  name: cluster-a
spec:
  metadataEndpoint: https://istio.k8s-a.example.com/metadata/
```

Cluster B:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioMulticluster
metadata:
  name: cluster-b
spec:
  metadataEndpoint: https://istio.k8s-b.example.com/metadata/
```

Additional resources

- Global custom resources
- Module `istio` documentation

Federation

Federation with Istio (Service Mesh)

Requirements for clusters

- Each cluster must have a unique domain in the `clusterDomain` parameter of the ClusterConfiguration resource. Note that none of the clusters should use the domain `cluster.local`, which is the default setting.

  > **i** `cluster.local` can't be used as it's an unmodified alias for the local cluster domain. When specifying `cluster.local` as a principals in the AuthorizationPolicy, it will always refer to the local cluster, even if there is another cluster in the mesh with `clusterDomain` explicitly defined as `cluster.local` (for details, refer to the Istio documentation).

- There are no requirements for unique subnets of modules and services in the `podSubnetCIDR` and `serviceSubnetCIDR` parameters of the resource ClusterConfiguration when clusters are operating in a federation.

  > **i** When analyzing traffic, Istio uses:
  >
  > - For HTTP and HTTPS requests — headers.
  > - For TCP requests — destination IP address and port number.
  >
  > Istio operates in the multi-network mode — pods from different clusters can only communicate with each other through the Istio ingress gateway. Direct communication between pods of different clusters is not supported.

General principles of federation

- Federation requires mutual trust between clusters. It requires a mutual root certificate exchange: cluster A must trust cluster B, and vice versa.
- Configuring inter-cluster access to services requires exchanging information about public services. To expose the bar service from cluster B to cluster A, you must create a ServiceEntry resource in cluster A, defining the Ingress gateway public address of cluster B.

Enabling the federation — created services

Enabling federation (via the `istio.federation.enabled = true` module parameter) results in the following:

- The `ingressgateway` service is added to the cluster. Its task is to proxy mTLS traffic coming from outside to application services.
- Another added service exports the following cluster metadata to the outside:
  - Istio root certificate (accessible without authentication).
  - List of public services in the cluster (available only for authenticated requests from neighboring clusters).
  - List of public addresses of the `ingressgateway` service (available only for authenticated requests from neighboring clusters).

Managing the federation

To establish a federation, you must:

- Create a set of IstioFederation resources in each cluster that describe all the other clusters.
  - After successful auto-negotiation between clusters, the IstioFederation resource will be filled with necessary service data in `status.metadataCache.public` and `status.metadataCache.private`.
- Add the `federation.istio.deckhouse.io/public-service: ""` label to each Service resource that is considered public within the federation.
  - In the other federation clusters, a corresponding ServiceEntry will be created for each Service, leading to the `ingressgateway` of the original cluster.

> **Important**. Ensure that the `name` field in the `.spec.ports` section of the Services resource is filled in for each port. Otherwise, there may be issues in the federation's work.

Example of configuring a federation of two clusters

To set up a federation with Istio, use the IstioFederation custom resource.

Cluster A:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioFederation
metadata:
  name: cluster-a
spec:
  metadataEndpoint: https://istio.k8s-a.example.com/metadata/
  trustDomain: cluster-a.local
```

Cluster B:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioFederation
metadata:
  name: cluster-b
spec:
  metadataEndpoint: https://istio.k8s-b.example.com/metadata/
  trustDomain: cluster-b.local
```

Additional resources

- Global custom resources
- Module `istio` custom resources

Network policy configuration

If there are requirements to restrict network interaction between servers at the infrastructure level where the Deckhouse Kubernetes Platform operates, the following conditions must be met:

- Tunneling mode for traffic between pods is enabled (configuration for CNI Cilium, configuration for CNI Flannel).
- Traffic between `podSubnetCIDR` encapsulated within a VXLAN is allowed (if inspection and filtering of traffic within a VXLAN tunnel is performed).
- If there is integration with external systems (such as LDAP, SMTP, or other external APIs), it is required to allow network communication with them.
- Local network communication is fully allowed within each individual cluster node.
- Inter-node communication is allowed on the ports shown in the platform component network interaction list. Note that most ports are in the 4200-4299 range. When new platform components are added, they will be assigned ports from this range (if it is possible).

Configuring network policies via standard Kubernetes means

Example network policy configuration

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

Configuring cluster-wide network policies using CiliumClusterwideNetworkPolicy

To define cluster-wide network policies in Deckhouse Kubernetes Platform, you can use the CiliumClusterwideNetworkPolicy objects of the `cni-cilium` module.

> ✖ Using CiliumClusterwideNetworkPolicies while the `policyAuditMode` option is not enabled in the `cni-cilium` module configuration may lead to incorrect operation of the control plane or loss of SSH access to all cluster nodes.

To use CiliumClusterwideNetworkPolicies, follow these steps:

1. Apply the primary set of CiliumClusterwideNetworkPolicy objects. To do this, in the `cni-cilium` module configuration, add the option `policyAuditMode` and set it to `true`.

   The `policyAuditMode` option can be removed after applying all CiliumClusterwideNetworkPolicy objects and verifying their functionality in Hubble UI.

2. Apply the network security policy rule:

```
apiVersion: "cilium.io/v2"
kind: CiliumClusterwideNetworkPolicy
metadata:
  name: "allow-control-plane-connectivity"
spec:
  ingress:
  - fromEntities:
    - kube-apiserver
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/control-plane: ""
```

If CiliumClusterwideNetworkPolicies are not used, the control plane may work incorrectly for up to a minute during the reboot of `cilium-agent` Pods. This occurs due to Conntrack table reset. Binding to the `kube-apiserver` entity helps to bypass the bug.

Additional resources

- Global custom resources
- Module `cni-cilium` documentation
- Module `cni-flannel` parameters

Software-defined networking (SDN)

Deckhouse Kubernetes Platform (DKP) supports adding additional software-defined networks (SDN) to the cluster. The functions of additional software-defined networking (hereinafter referred to as additional networks) within DKP are implemented using the `sdn` module.

Software-defined networking in DKP allow declarative management of additional network segments for application workloads (pods, virtual machines). Instead of manually configuring network interfaces on each cluster node, the administrator describes the desired network state through custom Kubernetes resources, and the `sdn` module automatically configures the necessary network equipment.

DKP supports the following features for working with software-defined networks:

- Configuration of network interfaces on nodes. Features such as port aggregation, bridging network interfaces, and configuring VLAN interfaces are supported.
- Additional networks for application workloads. Supports adding additional software-defined networks to the cluster: publicly available in each project (cluster) and available within a single namespace (project network).
- IPAM for additional networks. The mechanism allows you to allocate IP addresses (IPv4 addresses are supported) from pools and assign them to additional network interfaces of pods connected to cluster networks and project networks.
- Underlay networks for hardware device passthrough. This allows DPDK applications and other high-performance workloads to directly access physical network interfaces (PF/VF), bypassing the kernel network stack.
- System networks (service networks at nodes). These are designed for service traffic at the node level and are not used as additional pod networks. They are created on top of the existing underlay networks.

Additional resources

- Module `sdn` documentation

Configuring SDN in a cluster

To use SDN in a DKP cluster, you need to prepare the infrastructure for enabling the `sdn` module, as well as perform some preparatory actions after enabling it.

Preparing the infrastructure for enabling the `sdn` module

Before using additional software-defined networks (hereinafter referred to as additional networks) in a cluster, preliminary infrastructure preparation is required:

- **For creating additional networks based on tagged VLANs:**
  - Allocate VLAN ID ranges on the data center switches and configure them on the corresponding switch interfaces.
  - Select physical interfaces on the nodes for subsequent configuration of tagged VLAN interfaces. You can reuse interfaces already used by the DKP local network.

- **For creating additional networks based on direct, untagged access to a network interface:**
  - Reserve separate physical interfaces on the nodes and connect them into a single local network at the data center level.

Enabling the `sdn` module

Enable the `sdn` module according to the instructions.

Actions after enabling the sdn module

After enabling the module, NodeNetworkInterface objects will automatically appear in the cluster, reflecting the current state of the nodes.

To check for resources, use the command:

```
d8 k get nodenetworkinterface
NAME                           MANAGEDBY   NODE           TYPE     IFNAME           IFINDEX   STATE
AGE
virtlab-ap-0-nic-1c61b4a68c2a   Deckhouse   virtlab-ap-0   NIC      eth1             3         Up
35d
virtlab-ap-0-nic-fc34970f5d1f   Deckhouse   virtlab-ap-0   NIC      eth0             2         Up
35d
virtlab-ap-1-nic-1c61b4a6a0e7   Deckhouse   virtlab-ap-1   NIC      eth1             3         Up
35d
virtlab-ap-1-nic-fc34970f5c8e   Deckhouse   virtlab-ap-1   NIC      eth0             2         Up
35d
virtlab-ap-2-nic-1c61b4a6800c   Deckhouse   virtlab-ap-2   NIC      eth1             3         Up
35d
virtlab-ap-2-nic-fc34970e7ddb   Deckhouse   virtlab-ap-2   NIC      eth0             2         Up
35d
```

i When discovering node interfaces, the controller affixes the following labels, which are service labels (example):

```
labels:
  network.deckhouse.io/interface-mac-address: fa163eebea7b
  network.deckhouse.io/interface-type: NIC
  network.deckhouse.io/nic-pci-bus-info: 0000-17-00.0
  network.deckhouse.io/nic-pci-type: PF
  network.deckhouse.io/node-name: worker-01
annotations:
  network.deckhouse.io/heritage: NetworkController
```

In this example, each cluster node has two network interfaces: eth0 (DKP local network) and eth1 (dedicated interface for additional networks).

Marking interfaces for organizing additional software-defined networks

To enable the configuration of additional software-defined networks, label the dedicated interfaces that are planned to be used for network creation (in the example above, eth1) with an appropriate label:

```
d8 k label nodenetworkinterface virtlab-ap-0-nic-1c61b4a68c2a nic-group=extra
d8 k label nodenetworkinterface virtlab-ap-1-nic-1c61b4a6a0e7 nic-group=extra
d8 k label nodenetworkinterface virtlab-ap-2-nic-1c61b4a6800c nic-group=extra
```

Combining multiple physical interfaces into a channel aggregation interface (bond interface)

To increase bandwidth or redundancy, it is possible to combine several physical interfaces into a bond interface (channel aggregation interface).

i Only network interfaces located on the same physical or virtual host can be combined.

Example configuring a bond interface:

1. Set custom labels on the interfaces intended for aggregation.

   Example of setting the `nni.example.com/bond-group=bond0` label on interfaces:

   ```
   d8 k label nni node-0-nic-fa163efbde48 nni.example.com/bond-group=bond0
   d8 k label nni node-0-nic-fa40asdxzx78 nni.example.com/bond-group=bond0
   ```

2. Prepare the configuration for creating the interface and apply it.

   Configuration example:

   ```
   apiVersion: network.deckhouse.io/v1alpha1
   kind: NodeNetworkInterface
   metadata:
     name: nni-worker-01-bond0
   spec:
     nodeName: worker-01
     type: Bond
     heritage: Manual
     bond:
       bondName: bond0
       memberNetworkInterfaces:
         - labelSelector:
             matchLabels:
               # This is a service label that needs to be combined with the Bond
   interface on a specific node.
               network.deckhouse.io/node-name: worker-01
               # Custom label (was added to the interfaces in the previous step).
               nni.example.com/bond-group: bond0
   ```

3. Check the status of the created Bond interface:

   Get a list of interfaces:

   ```
   d8 k get nni
   ```

   Example output:

   ```
   NAME                    MANAGEDBY   NODE                                    TYPE
   IFNAME      IFINDEX   STATE   AGE
   nni-worker-01-bond0     Manual      worker-01-b23d3a26-5fb4b-5s9fp      Bond
   bond0       76        Up      7m48s
   ...
   ```

   Check the status of the desired interface:

   ```
   d8 k get nni nni-worker-01-bond0 -o yaml
   ```

   Example of interface status:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: NodeNetworkInterface
metadata:
...
status:
  conditions:
  - lastProbeTime: "2025-09-30T09:00:54Z"
    lastTransitionTime: "2025-09-30T09:00:39Z"
    message: Interface created
    reason: Created
    status: "True"
    type: Exists
  - lastProbeTime: "2025-09-30T09:00:54Z"
    lastTransitionTime: "2025-09-30T09:00:39Z"
    message: Interface is up and ready to send packets
    reason: Up
    status: "True"
    type: Operational
  deviceMAC: 6a:c7:ab:2a:a6:1e
  groupedLinks:
  - deviceMAC: fa:16:3e:92:14:40
    type: NIC
  ifIndex: 76
  ifName: bond0
  managedBy: Manual
  operationalState: Up
  permanentMAC: ""
```

Configuring and connecting additional virtual networks for use in application pods

The Deckhouse Kubernetes Platform provides the ability to declaratively manage additional networks for application workloads (pods, virtual machines). At the same time:

- Each additional network implies a single L2 data exchange domain.
- Within the Pod’s network namespace, an additional network is represented as a tap interface.
- The following modes are currently available for L2 network implementation:
  - **Tagged VLAN**: Communication between Pods on different Nodes uses VLAN-tagged packets and the infrastructure’s network equipment for switching. This method allows to create up to 4096 additional networks within a single cluster.
  - **Direct access to a Node’s network interface**: Communication between Pods on different Nodes uses the local network interfaces of the Nodes.
- From a network management perspective, there are two types of networks:
  - **Cluster network**: A network available in all projects, under administrator management. Example: a public WAN network or a shared network for cross-project traffic.
  - **Project network**: A network available within a Namespace, under user management.

Custom resources ClusterNetwork, Network, and NetworkClass are used to configure and connect additional networks for application pods.

> **i** If the VLAN type was specified in the Network or ClusterNetwork resources, NodeNetworkInterface will also be created for VLAN and Bridge.

> ⚠ Before creating an additional network, mark the interfaces that will be used by it.

Creating a publicly accessible network (cluster)

A custom resource ClusterNetwork is used to create publicly accessible networks across the entire cluster.

Creating a network based on tagged traffic

To create a network based on tagged traffic, follow these steps:

1. Create and apply the ClusterNetwork resource:

   In the `spec.type` parameter, specify the value `VLAN`. Tagged interfaces will be configured on the corresponding network interfaces of the nodes to ensure connectivity via the VLAN provided by the infrastructure.

   Example of a ClusterNetwork manifest for creating a public network based on tagged traffic:

   ```
   apiVersion: network.deckhouse.io/v1alpha1
   kind: ClusterNetwork
   metadata:
     name: my-cluster-network
   spec:
     type: VLAN
     vlan:
       id: 900
     parentNodeNetworkInterfaces:
       labelSelector:
         matchLabels:
           # Manually applied label on NodeNetworkInterface resources.
           nic-group: extra
   ```

2. Check the status of the created resource with the command:

   ```
   d8 k get clusternetworks.network.deckhouse.io my-cluster-network -o yaml
   ```

   Example of the status of a ClusterNetwork resource:

   ```
   apiVersion: network.deckhouse.io/v1alpha1
   kind: ClusterNetwork
   metadata:
   ...
   status:
     bridgeName: d8-br-900
     conditions:
     - lastTransitionTime: "2025-09-29T14:39:20Z"
       message: All node interface attachments are ready
       reason: AllNodeInterfaceAttachmentsAreReady
       status: "True"
       type: AllNodeAttachementsAreReady
     - lastTransitionTime: "2025-09-29T14:39:20Z"
       message: Network is operational
       reason: NetworkReady
       status: "True"
       type: Ready
     nodeAttachementsCount: 1
     observedGeneration: 1
     readyNodeAttachementsCount: 1
   ```

3. Check the connection of the additional network to the interfaces on the nodes .

Creating a network based on direct interface access

To create a network based on direct interface access, use the ClusterNetwork resource. In the `spec.type` parameter, specify the value `Access`. The corresponding network adapters on the nodes will be used directly to provide connectivity.

Example manifest for a network based on direct interface access:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterNetwork
metadata:
  name: my-cluster-network
spec:
  type: Access
  parentNodeNetworkInterfaces:
    labelSelector:
      matchLabels:
        # Manually applied label on NodeNetworkInterface resources.
        nic-group: extra
```

After creating the network, check the connection of the additional network to the interfaces on the nodes.

Creating a project network (user network)

In order for users to be able to create their own dedicated networks based on tagged traffic, it is necessary to first describe the range of tags available to them and define the network interfaces on which they can be configured. To do this, use the custom resource NetworkClass.

Example:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: NetworkClass
metadata:
  name: my-network-class
spec:
  vlan:
    idPool:
    - 600-800
    - 1200
    parentNodeNetworkInterfaces:
      labelSelector:
        matchLabels:
          nic-group: extra
```

Upon user request, the administrator provides them with the name of the created NetworkClass, which is used when creating the project network.

An example of creating a custom network using the NetworkClass resource administrator is described in the section “Creating a project network (user network)”.

Checking the connection of an additional network to interfaces on nodes

After creating ClusterNetwork or Network, the controller will create a NodeNetworkInterfaceAttachment tracking resource to link it to a NodeNetworkInterface.

To obtain a list of NodeNetworkInterfaceAttachment resources and information about a specific resource, use the following commands:

```
d8 k get nnia
d8 k get nnia my-cluster-network-... -o yaml
```

Example NodeNetworkInterfaceAttachment resource:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: NodeNetworkInterfaceAttachment
metadata:
...
  finalizers:
    - network.deckhouse.io/nni-network-interface-attachment
    - network.deckhouse.io/pod-network-interface-attachment
  generation: 1
  name: my-cluster-network-...
...
spec:
  networkRef:
    kind: ClusterNetwork
    name: my-cluster-network
  parentNetworkInterfaceRef:
    name: right-worker-b23d3a26-5fb4b-h2bkv-nic-fa163eebea7b
  type: VLAN
status:
  bridgeNodeNetworkInterfaceName: right-worker-b23d3a26-5fb4b-h2bkv-bridge-900
  conditions:
    - lastTransitionTime: "2025-09-29T14:39:06Z"
      message: Vlan created
      reason: VLANCreated
      status: "True"
      type: Exist
    - lastTransitionTime: "2025-09-29T14:39:06Z"
      message: Bridged successfully
      reason: VLANBridged
      status: "True"
      type: Ready
  nodeName: right-worker-b23d3a26-5fb4b-h2bkv
  vlanNodeNetworkInterfaceName: right-worker-b23d3a26-5fb4b-h2bkv-vlan-900-60f3dc
```

The NodeNetworkInterfaceAttachment status will change to `True` immediately after the corresponding NodeNetworkInterface appears and transitions to the `Up` state.

To check the status of NodeNetworkInterface, use the command:

```
d8 k get nni
```

Example output:

```
NAME                                                MANAGEDBY   NODE
TYPE     IFNAME      IFINDEX   STATE   AGE
...
right-worker-b23d3a26-5fb4b-h2bkv-bridge-900        Deckhouse   right-worker-
b23d3a26-5fb4b-h2bkv   Bridge   d8-br-900   684       Up      14h
right-worker-b23d3a26-5fb4b-h2bkv-nic-fa163eebea7b  Deckhouse   right-worker-
b23d3a26-5fb4b-h2bkv   NIC      ens3        2         Up      19d
right-worker-b23d3a26-5fb4b-h2bkv-vlan-900-60f3dc   Deckhouse   right-worker-
b23d3a26-5fb4b-h2bkv   VLAN     ens3.900    683       Up      14h
...
```

IPAM: IP address pools for additional networks

The IPAM mechanism allows you to automatically allocate and assign **IPv4 addresses** for additional network interfaces of pods connected to cluster networks and project networks.

Principles and features of IPAM in DKP

For each required IP address, a object IPAddress (ClusterIPAddress — for cluster networks) is created and used, which references the project network or cluster network. The controller allocates an address

from the pool and stores the result in `status.address`, `status.network`, `status.routes` of the IPAddress (ClusterIPAddress) object. The agent on the node assigns an IP address and routes to the interface inside the pod and sets the `status.conditions[Attached]` and `status.usedByPods` fields of the IPAddress (ClusterIPAddress) object.

Protection against conflicts when reusing IP addresses

To protect against conflicts, a cluster-scoped object IPAddressLease is created, which reserves the IP address. When an IPAddress object is deleted, the corresponding IPAddressLease is marked as `orphaned` (using the `status.orphaningTimestamp` field) and holds the address for the time specified in the `spec.ttl` parameter (to avoid rapid reuse).

Resources and settings for configuring IPAM

The following are used to manage the allocation and assignment of IP addresses:

- Address pools: for cluster networks (resource ClusterIPAddressPool) or project networks (resource IPAddressPool).
- Parameters for enabling IPAM for a specific network: `Network.spec.ipam.ipAddressPoolRef` — for project networks, `ClusterNetwork.spec.ipam.ipAddressPoolRef` — for cluster networks.
- The IPAddress resource is a request (automatic or static) for an address allocation, which is then assigned to the pod interface.

Example of assigning IP addresses to the cluster network

> **i** To allocate an address pool for a cluster network, use the ClusterIPAddressPool resource.

To allocate a pool of addresses and assign them to network interfaces of pods connected to the cluster network, perform the following steps:

1. Create an address pool. To do this, use the ClusterIPAddressPool resource.

   Example:

   ```
   apiVersion: network.deckhouse.io/v1alpha1
   kind: ClusterIPAddressPool
   metadata:
     name: public-net-pool
   spec:
     leaseTTL: 24h
     pools:
       - network: 203.0.113.0/24
         ranges:
           - 203.0.113.10-203.0.113.200
   ```

   > **i** The `spec.pools[].ranges` parameter is optional. If it is not specified, the entire CIDR from `spec.pools[].network` is considered available (except for network/broadcast addresses, see the behavior of `/31` and `/32`).

2. Enable IPAM on the network. To do this, specify the parameters of the lusterIPAddressPool created in the previous step in the `spec.ipam.ipAddressPoolRef` parameter of the ClusterNetwork resource:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterNetwork
metadata:
  name: my-cluster-network
spec:
  type: VLAN
  vlan:
    id: 900
  parentNodeNetworkInterfaces:
    labelSelector:
      matchLabels:
        nic-group: extra
  ipam:
    ipAddressPoolRef:
      kind: ClusterIPAddressPool
      name: public-net-pool
```

After allocating a pool of IP addresses for the cluster network, they can be assigned to the network interfaces of pods connected to this network. For more details, see the section Assigning IP addresses to network interfaces of pods connected to an additional network.

Configuring and connecting underlay networks for hardware device passthrough

This feature is designed for high-performance workloads that require direct access to hardware, such as DPDK applications.

Key features

DKP implements the following features for working with underlay networks:

- **Hardware device passthrough**: Physical network interfaces (PF/VF) are directly exposed to pods, bypassing the kernel network stack for maximum performance.
- **SR-IOV configuration**: Automatic configuration of SR-IOV on selected Physical Functions to create Virtual Functions, allowing multiple pods to share the same hardware.
- **DPDK support**: Devices can be bound in different modes suitable for DPDK workloads:
  - **VFIO-PCI**: Explicitly connects a network device to the pod by binding it to the `vfio-pci` driver. The corresponding VFIO device files (e.g., `/dev/vfio/vfio0`) are mounted into the pod for userspace access.
  - **DPDK**: A universal mode that automatically selects the appropriate driver for the network adapter vendor. For Mellanox NICs, the device is bound to the `mlx5_core` driver with both the netdev interface and necessary device files mounted (InfiniBand verbs files, `/dev/net/tun`, and the corresponding sysfs directory). For other vendors, the device is bound via VFIO (same as VFIO-PCI mode).
  - **NetDev**: Only the Linux network interface is passed through to the pod as a standard kernel network device.

Operation modes

The following device allocation modes are supported, which determine how physical interfaces are provided to hosts:

- **Shared mode**: Creates Virtual Functions (VF) from Physical Functions (PF) using SR-IOV, allowing multiple pods to share the same hardware. Each pod receives one or more VFs.
- **Dedicated mode**: Exposes each Physical Function as an exclusive device without SR-IOV. Each pod gets exclusive access to a complete PF, suitable for workloads requiring maximum performance.

Automatic interface grouping

When `autoBonding` is enabled, the controller groups interfaces from multiple matched PFs into a single DRA device. The interfaces are passed through to the pod as separate network interfaces, allowing applications (e.g., DPDK) to handle bonding/aggregation at the application level. Note that this does not create kernel-level bonding interfaces inside the pod.

Procedure for configuring and connecting physical interfaces to application subnets

To create underlay networks for forwarding hardware devices to pods, a custom resource UnderlayNetwork is used. It provides direct connection of physical network interfaces (Physical Functions and Virtual Functions) to pods via Kubernetes Dynamic Resource Allocation (DRA).

Prerequisites for DPDK applications

Before configuring UnderlayNetwork resources, you must prepare the cluster's worker nodes for DPDK applications:

- Configure hugepages.
- Configure Topology Manager.

Configuring hugepages

DPDK applications require hugepages for efficient memory management. Configure hugepages on all worker nodes using NodeGroupConfiguration:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: hugepages-for-dpdk
spec:
  bundles:
    - "*"  # Apply to all OS bundles.
  nodeGroups:
    - "*"  # Apply to all node groups.
  weight: 100
  content: |
    #!/bin/bash
    echo "vm.nr_hugepages = 4096" > /etc/sysctl.d/99-hugepages.conf
    sysctl -p /etc/sysctl.d/99-hugepages.conf
```

This configuration sets `vm.nr_hugepages = 4096` on all nodes, providing 8 GiB of hugepages (4096 pages × 2 MiB per page).

Configuring Topology Manager

For optimal performance, enable Topology Manager on NodeGroups of worker nodes where DPDK applications will run. This ensures that CPU, memory, and device resources are allocated from the same NUMA node.

Example NodeGroup configuration:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  kubelet:
    topologyManager:
      enabled: true
      policy: SingleNumaNode
      scope: Container
  nodeType: Static
```

For more information, see:

- topologyManager.enabled
- topologyManager.policy.

Preliminary steps before creating an UnderlayNetwork

Before creating an UnderlayNetwork, ensure that:

1. Physical network interfaces (NICs) are available on the nodes and are discovered as

NodeNetworkInterface resources.
2. The interfaces you plan to use are Physical Functions (PF), not Virtual Functions (VF).
3. For Shared mode, the NICs must support SR-IOV.

Checking and configuring network interfaces (NodeNetworkInterface)

First, check which Physical Functions are available on your nodes:

```
d8 k get nni -l network.deckhouse.io/nic-pci-type=PF
```

Example output:

```
NAME                            MANAGEDBY   NODE          TYPE   IFNAME   IFINDEX
STATE   VF/PF   Binding   Driver      Vendor   AGE
worker-01-nic-0000:17:00.0      Deckhouse   worker-01     NIC    ens3f0   3
Up      PF      NetDev    ixgbe       Intel    35d
worker-01-nic-0000:17:00.1      Deckhouse   worker-01     NIC    ens3f1   4
Up      PF      NetDev    ixgbe       Intel    35d
worker-02-nic-0000:17:00.0      Deckhouse   worker-02     NIC    ens3f0   3
Up      PF      NetDev    ixgbe       Intel    35d
worker-02-nic-0000:17:00.1      Deckhouse   worker-02     NIC    ens3f1   4
Up      PF      NetDev    ixgbe       Intel    35d
```

Label the interfaces that will be used for UnderlayNetwork:

```
d8 k label nni worker-01-nic-0000:17:00.0 nic-group=dpdk
d8 k label nni worker-01-nic-0000:17:00.1 nic-group=dpdk
d8 k label nni worker-02-nic-0000:17:00.0 nic-group=dpdk
d8 k label nni worker-02-nic-0000:17:00.1 nic-group=dpdk
```

i You can check the PCI information and SR-IOV support status for each interface:

```
d8 k get nni worker-01-nic-0000:17:00.0 -o json | jq '.status.nic.pci.pf'
```

The `status.nic.pci.pf.sriov.supported` section contains information about SR-IOV support.

Creating UnderlayNetwork in Dedicated mode

In Dedicated mode, each Physical Function is exposed as an exclusive device. This mode is suitable when:

- SR-IOV is not available or not needed.
- Each pod needs exclusive access to a complete PF.

To create an Underlay network in Dedicated mode, follow these steps:

1. Create and apply the UnderlayNetwork resource. In the `spec.mode` field, specify the value `Dedicated`.

   Example configuration:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: UnderlayNetwork
metadata:
  name: dpdk-dedicated-network
spec:
  mode: Dedicated
  autoBonding: false
  memberNodeNetworkInterfaces:
    - labelSelector:
       matchLabels:
         nic-group: dpdk # Label used to mark interfaces during the verification
and configuration of network interfaces.
```

When `autoBonding` is set to `true`, all matched PFs on a node are grouped into a single DRA device, exposing all PFs to the pod as separate interfaces. When `false`, each PF is published as a separate DRA device.

2. Check the status of the created UnderlayNetwork:

   ```
   d8 k get underlaynetwork dpdk-dedicated-network -o yaml
   ```

   Example status of UnderlayNetwork in `Dedicated` mode:

   ```
   apiVersion: network.deckhouse.io/v1alpha1
   kind: UnderlayNetwork
   metadata:
     name: dpdk-dedicated-network
   ...
   status:
     observedGeneration: 1
     conditions:
     - message: All 2 member node network interface selectors have matches
       observedGeneration: 1
       reason: AllInterfacesAvailable
       status: "True"
       type: InterfacesAvailable
   ```

Creating UnderlayNetwork in Shared mode

In `Shared` mode, Virtual Functions (VF) are created from Physical Functions (PF) using SR-IOV, allowing multiple pods to share the same hardware. This mode requires SR-IOV support on the NICs.

To create an Underlay network in `Shared` mode, follow these steps:

1. Create and apply the UnderlayNetwork resource. In the `spec.mode` field, specify the value `Shared`.

   Example configuration:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: UnderlayNetwork
metadata:
  name: dpdk-shared-network
spec:
  mode: Shared
  autoBonding: true
  memberNodeNetworkInterfaces:
    - labelSelector:
        matchLabels:
          nic-group: dpdk
  shared:
    sriov:
      enabled: true
      numVFs: 8
```

In this example:

- `mode: Shared` enables SR-IOV and VF creation.
- `autoBonding: true` groups one VF from each matched PF into a single DRA device.
- `shared.sriov.enabled: true` enables SR-IOV on selected PFs.
- `shared.sriov.numVFs: 8` creates 8 Virtual Functions per Physical Function.

> **i** The `mode` and `autoBonding` fields are immutable once set. Plan your configuration carefully before creating the resource.

2. After creating the UnderlayNetwork, monitor the SR-IOV configuration status:

```
d8 k get underlaynetwork dpdk-shared-network -o yaml
```

Example status of UnderlayNetwork in `Shared` mode:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: UnderlayNetwork
metadata:
  name: dpdk-shared-network
...
status:
  observedGeneration: 1
  sriov:
    supportedNICs: 4
    enabledNICs: 4
  conditions:
  - lastTransitionTime: "2025-01-15T10:30:00Z"
    message: SR-IOV configured on 4 NICs
    reason: SRIOVConfigured
    status: "True"
    type: SRIOVConfigured
  - lastTransitionTime: "2025-01-15T10:30:05Z"
    message: Interfaces are available for allocation
    reason: InterfacesAvailable
    status: "True"
    type: InterfacesAvailable
```

3. Verify that VFs have been created by checking NodeNetworkInterface resources:

```
d8 k get nni -l network.deckhouse.io/nic-pci-type=VF
```

Preparing namespaces for UnderlayNetwork usage

Before users can request UnderlayNetwork devices in their pods, the namespace must be labeled to enable UnderlayNetwork support. This is an administrative task that should be done for namespaces where DPDK applications will run:

```
d8 k label namespace mydpdk direct-nic-access.network.deckhouse.io/enabled=""
```

Configuring and connecting system networks (service networks)

System networks (service networks) are intended for internal traffic at the node level (e.g., for storage, management, etc.) and are not used as additional pod networks.

Additional service networks are created on cluster nodes on top of existing underlay networks. The custom resource SystemNetwork is used for this purpose. System networks obtain IP addresses from the ClusterIPAddressPool during IPAM configuration.

Principles and features of system networks:

- **Working on top of underlay networks**: A system network attaches to an underlay network (UnderlayNetwork). To connect, specify the name of the underlay network to which the system network should be connected in the `spec.underlayNetworkName` parameter of the SystemNetwork resource. The set of node interfaces (PF or VF) used by the system network is defined in the `memberNodeNetworkInterfaces` parameter of the UnderlayNetwork resource.
- **Support for different types of system network connections to the underlay network**: You can create a VLAN sub-interface (`type: VLAN`), use the underlay as-is (`type: Access`), or attach via an SR-IOV Virtual Function (`type: SRIOVVirtualFunction`) with optional tuning (MTU, MAC, spoof checking, etc.).
- **Support for IPAM mechanism**: Optional `spec.ipam` parameter references a ClusterIPAddressPool. The controller and agent then allocate addresses from that pool and assign them to the node interfaces for this system network.
- **Tracking the status of system networks**: The agent reports node addresses (including SystemNetwork IPs) in NodeNetworkStatus resources. Internal SystemNetworkNodeNetworkInterfaceAttachment resources track the binding of each system network to each node’s parent interface.

Prerequisites for creating and using system networks

To enable system networks in a cluster, the following requirements must be met:

1. An underlay network must exist. The system network will attach to it by name via `spec.underlayNetworkName` parameter. The underlay’s `memberNodeNetworkInterfaces` selectors determine which node interfaces are used.
2. **Optional**. To automatically assign IP addresses on interfaces belonging to a system network, create a system network address pool. It must be specified in the `spec.ipam.clusterIPAddressPoolName` parameter of the SystemNetwork resource when the network is created.

Creating a system network

To create a system network, use the SystemNetwork resource. Supported system networks have the following types of connections to the underlay networks on which they will operate:

- `VLAN`: VLAN interfaces are created on interfaces connected to the underlay network (underlay interfaces).
- `Access`: Direct access (without VLAN) to interfaces on nodes connected to the underlay network.
- `SRIOVVirtualFunction`: Connection to a physical interface via SR-IOV.

VLAN type

Creates a VLAN sub-interface on each matched underlay interface. When creating a network, you must specify the VLAN ID (parameter `spec.vlan.id`).

Example of a system network manifest with a `VLAN` connection type to the underlay network:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: SystemNetwork
metadata:
  name: storage-network
spec:
  type: VLAN
  underlayNetworkName: my-underlay
  vlan:
    id: 100
  ipam:
    clusterIPAddressPoolName: storage-pool
```

To check the network status after creation, use the section Checking system network status.

Access type

Uses the underlay interface as-is (no VLAN tag). Useful when the underlay already represents a single L2 segment.

Example of a system network manifest with a `Access` connection type to the underlay network:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: SystemNetwork
metadata:
  name: mgmt-network
spec:
  type: Access
  underlayNetworkName: my-underlay
  ipam:
    clusterIPAddressPoolName: mgmt-pool
```

To check the network status after creation, use the section Checking system network status.

SRIOVVirtualFunction type

Attaches via an SR-IOV Virtual Function. The underlay must be in `Shared` mode so that VFs exist. You can optionally tune VF settings.

Example of a system network manifest with a `SRIOVVirtualFunction` connection type to the underlay network:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: SystemNetwork
metadata:
  name: vf-service-network
spec:
  type: SRIOVVirtualFunction
  underlayNetworkName: dpdk-shared-network
  sriovVirtualFunction:
    vlan: 200
    mtu: 1500
    linkState: Auto
  ipam:
    clusterIPAddressPoolName: vf-pool
```

To check the network status after creation, use the section Checking system network status.

Creating a pool of IP addresses for configuring the IPAM system network

To create an address pool for configuring the system network IPAM, use the ClusterIPAddressPool resource:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ClusterIPAddressPool
metadata:
  name: storage-pool
spec:
  leaseTTL: 24h
  pools:
    - network: 10.20.30.0/24
      ranges:
        - 10.20.30.10-10.20.30.250
```

When creating a system network, specify this pool in the `spec.ipam.clusterIPAddressPoolName` parameter of the SystemNetwork resource.

Checking system network status

To get a list of system networks, use the command:

```
d8 k get systemnetworks
```

To view the status of a specific system network, use the command:

```
d8 k get systemnetwork storage-network -o yaml
```

In `status` you will see:

- `nodeAttachementsCount`: Total attachments (one per matched node interface).
- `readyNodeAttachementsCount`: Attachments that are configured and ready.
- `conditions` — e.g. `Ready`: When all attachments are ready.

To view internal attachments (one per a “system network + parent interface” pair), use the command:

```
d8 k get systemnetworknodenetworkinterfaceattachments
```

To view IP addresses at the node level (including system network IPs) for all nodes, use the command (one NodeNetworkStatus is assigned to each node):

```
d8 k get nodenetworkstatus
```

To view information about IP addresses at the node level (including system network IPs) for a specific node, use the command:

```
d8 k get nodenetworkstatus -l network.deckhouse.io/node-name=worker-01 -o yaml
```

In `status.addresses` look for entries with `type: SystemNetworkIP` and `systemNetworkName` set to your system network name.

Additional resources

- Module `node-manager` custom resources
- Module `sdn` documentation

Managing static routes and ip rules on cluster nodes

In Deckhouse Kubernetes Platform, static-routing and ip rule management on cluster nodes is implemented using the `static-routing-manager` module.

Configuration examples

Creating a route in main routing table

```
apiVersion: network.deckhouse.io/v1alpha1
kind: RoutingTable
metadata:
  name: myrt-main
spec:
  ipRoutingTableID: 254 # Main routing table ID is 254.
  routes:
  - destination: 10.0.0.0/8
    gateway: 192.168.0.1
  nodeSelector:
    node-role.deckhouse.io: load-balancer
```

Based on this configuration, the route `10.0.0.0/8 via 192.168.0.1` will be created on the nodes falling under `nodeSelector`:

```
ip -4 route ls
```

Example output:

```
10.0.0.0/8 via 192.168.0.1 dev eth0 realm 216
```

The `realm 216` instruction in the route is used as a marker to identify the route under module control (d8 hex = 216 dec).

Creating a route in additional routing table

```
apiVersion: network.deckhouse.io/v1alpha1
kind: RoutingTable
metadata:
  name: myrt-extra
spec:
  routes:
  - destination: 0.0.0.0/0
    gateway: 192.168.0.1
  nodeSelector:
    node-role.deckhouse.io: load-balancer
status:
  ipRoutingTableID: 10000 # If spec.ipRoutingTableID is not specified, it will be
generated automatically and placed in status.
```

According to this configuration, the route `10.0.0.0/8 via 192.168.0.1` will be created on the nodes falling under `nodeSelector` in the table `10000`:

```
ip -4 route ls table 10000
```

Example output:

```
default via 192.168.0.1 dev eth0 realm 216
```

Creating an ip rule

```
apiVersion: network.deckhouse.io/v1alpha1
kind: IPRuleSet
metadata:
  name: myiprule
spec:
  rules:
  - selectors:
      from:
      - 192.168.111.0/24
      - 192.168.222.0/24
      to:
      - 8.8.8.8/32
      - 172.16.8.0/21
      sportRange:
        start: 100
        end: 200
      dportRange:
        start: 300
        end: 400
      ipProto: 6
    actions:
      lookup:
        routingTableName: myrt-extra
    priority: 50
  nodeSelector:
    node-role.deckhouse.io: load-balancer
```

According to this configuration, an ip rule will be created on the nodes falling under `nodeSelector`. To view results, run:

```
ip rule list
```

Example output:

```
...
50: from 192.168.111.0/24 to 172.16.8.0/21 ipproto tcp sport 100-200 dport 300-400
lookup 10000 realms 216
50: from 192.168.222.0/24 to 8.8.8.8 ipproto tcp sport 100-200 dport 300-400 lookup
10000 realms 216
50: from 192.168.222.0/24 to 172.16.8.0/21 ipproto tcp sport 100-200 dport 300-400
lookup 10000 realms 216
50: from 192.168.111.0/24 to 8.8.8.8 ipproto tcp sport 100-200 dport 300-400 lookup
10000 realms 216
...
```

Additional resources

- Module `static-routing-manager` documentation

Managing DNS in a Kubernetes cluster

DNS management in a Kubernetes cluster is implemented using the `kube-dns` module.

The module installs CoreDNS components for managing DNS in the Kubernetes cluster.

> **i** The module deletes Deployments, ConfigMaps as well as RBAC for CoreDNS that were previously created using the `kubeadm` tool. When deploying your own CoreDNS, avoid using the names `coredns` or `system:coredns` for any resources (Deployment, Service, ConfigMap, ServiceAccount, ClusterRole, ClusterRoleBinding). Use alternative names like `infra-dns` to prevent automatic removal by Deckhouse Kubernetes Platform.

Configuration example

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  version: 1
  enabled: true
  settings:
    upstreamNameservers:
    - 8.8.8.8
    - 8.8.4.4
    hosts:
    - domain: one.example.com
      ip: 192.168.0.1
    - domain: two.another.example.com
      ip: 10.10.0.128
    stubZones:
    - zone: consul.local
      upstreamNameservers:
      - 10.150.0.1
    enableLogs: true
    clusterDomainAliases:
    - foo.bar
    - baz.qux
```

Additional resources

- Module `kube-dns` documentation

Caching DNS requests on cluster nodes

In Deckhouse Kubernetes Platform, you can deploy a local caching DNS server on each cluster node. It exports metrics to Prometheus for visualization in a Grafana dashboard.

This feature is implemented by the `node-local-dns` module. The module consists of the original CoreDNS deployed as a DaemonSet on all cluster nodes, along with a network configuration algorithm and iptables rules.

Detailed information about the problems that caching DNS-server allows to solve and the principle of its work is available in Caching DNS server in a cluster.

Example custom DNS configuration in a Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: dns-example
spec:
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
      - 169.254.20.10
  containers:
    - name: test
      image: nginx
```

For details on DNS configuration, refer to the Kubernetes documentation.

Additional resources

- Module `node-local-dns` documentation

BpfLB operation modes for external traffic processing

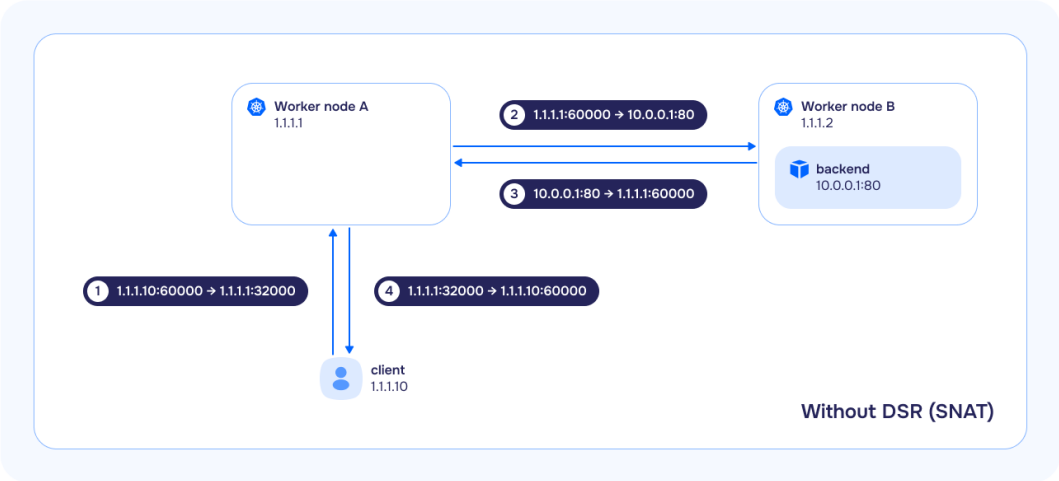
You can use the `cni-cilium` module to configure the BPF Load Balancer (bpfLB) mode of operation in Deckhouse Kubernetes Platform.

In Kubernetes, traffic typically comes through a load balancer that distributes incoming requests among servers and processes responses. Thus, the total throughput is limited by the load balancer resources.

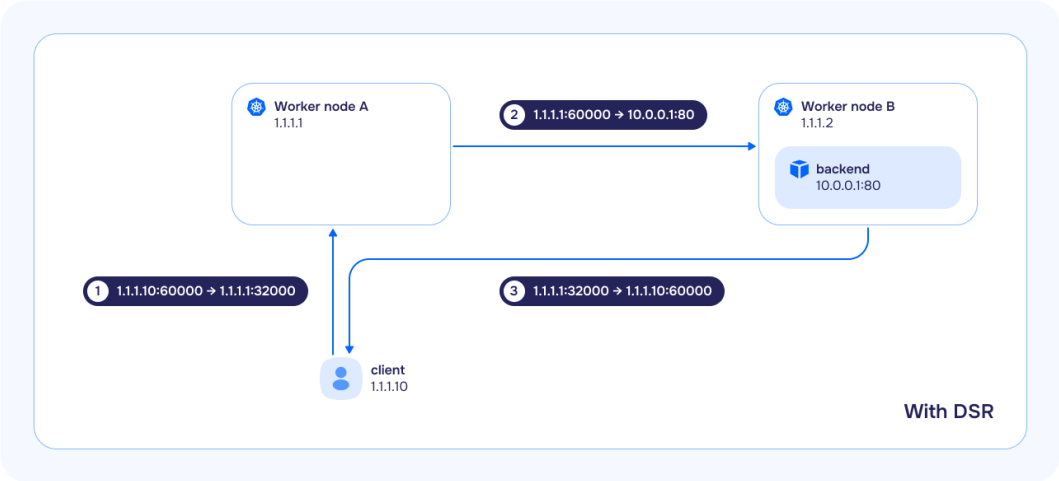
To optimize traffic and unload the balancer, the `DSR` mechanism is used, in which incoming packets go through the load balancer, and outgoing ones go directly from the terminating servers. Since responses are usually much larger in size than requests, this approach can significantly increase the overall throughput.

The module allows selecting the mode of operation via the `bpfLBMode` parameter, which affects the behavior of Services of the `NodePort` and `LoadBalancer` types:

- `SNAT` (Source Network Address Translation): A subtype of NAT in which, for each outgoing packet, the source IP address is translated to the IP address of the gateway from the target subnet, and incoming packets passing through the gateway are translated back based on a translation table. In this mode, `bpfLB` fully replicates the logic of `kube-proxy`:
  - If `externalTrafficPolicy: Local` is specified for a Service object, the traffic will be forwarded and balanced only to those target pods running on the same node where the traffic arrived. If the target pod is not running on this node, the traffic will be dropped.
  - If `externalTrafficPolicy: Cluster` is specified for a Service object, the traffic will be forwarded and balanced to all target pods in the cluster. At the same time, if the target pods are located on other nodes, SNAT will be performed when transmitting traffic to them (the source IP address will be replaced with the InternalIP of the node).



- `DSR` (Direct Server Return): A method where all incoming traffic passes through the load balancer, and all outgoing traffic bypasses it. This method is used instead of `SNAT`. Often, responses are much larger than requests, and `DSR` can significantly increase the overall throughput of the scheme:
  - If `externalTrafficPolicy: Local` is specified in the `Service`, its behavior is completely analogous to `kube-proxy` and `bpfLB` in `SNAT` mode.
  - If `externalTrafficPolicy: Cluster` is specified in the `Service`, the traffic will be forwarded and balanced to all target pods in the cluster. Note the following:
    - If the target pods are on other nodes, then the source IP address will be preserved when incoming traffic is sent to them.
    - Outgoing traffic will go directly from the node on which the target pod was launched.
    - The source IP address will be replaced with the external IP address of the node to which the incoming request came originally.



> **i** In case of using `DSR` and `Service` mode with `externalTrafficPolicy: Cluster` additional network environment settings are required.
> Network equipment must be ready for asymmetric traffic flow: IP address anti-spoofing tools (`uRPF`, `sourceGuard`, etc.) must be disabled or configured accordingly.

- `Hybrid`: In this mode, TCP traffic is handled per `DSR` rules, while UDP traffic is handled per `SNAT` rules.

Example of bpfLB operation mode configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  settings:
    tunnelMode: VXLAN
    bpfLBMode: SNAT # SNAT mode is selected.
  version: 1
  enabled: true
```

Additional resources

- Module `cni-cilium` documentation

Synchronizing time on nodes

To synchronize time on Deckhouse cluster nodes, DKP uses a built-in solution based on chrony. Using the Network Time Protocol (NTP), DKP ensures that system clocks on cluster nodes are synchronized with external NTP servers. If required, you can disable this built-in mechanism and configure custom NTP daemons.

Enabling built-in time synchronization

Enable the `chrony` module to activate time synchronization:

```
d8 system module enable chrony
```

By default, the time source is the server `pool.ntp.org`. You can specify a list of NTP servers using the `ntpServers` parameter in the configuration of the `chrony` module.

An example of the module configuration specifying NTP servers:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: chrony
spec:
  enabled: true
  settings:
    ntpServers:
      - ntp.ubuntu.com
      - time.google.com
  version: 1
```

Using custom NTP daemons

To disable the built-in time synchronization mechanism and use your own NTP daemons on the nodes, follow these steps:

1. Disable the `chrony` module:

   ```
   d8 system module disable chrony
   ```

   If the command is successful, you should see the message confirming that the module has been disabled:

   ```
   Module chrony disabled
   ```

2. Create a NodeGroupConfiguration resource to enable the NTP daemons on the nodes.

   Example for `systemd-timesyncd`:

   ```
   d8 k apply -f - <<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: NodeGroupConfiguration
   metadata:
     name: enable-ntp-on-node.sh
   spec:
     weight: 100
     nodeGroups: ["*"]
     bundles: ["*"]
     content: |
       systemctl enable systemd-timesyncd
       systemctl start systemd-timesyncd
   EOF
   ```

Additional resources

- Module `chrony` documentation
- Module `node-manager` custom resources

Access via OpenVPN

This section describes how to configure access to cluster resources using OpenVPN with certificate-based authentication.

Settings are available through a simple web interface that allows you to:

- Issue certificates;
- Revoke certificates;
- Undo certificate revocation;
- Download a ready-to-use user configuration file.

OpenVPN is integrated with user authentication, enabling access control for the web interface.

Configuration

External access to OpenVPN

You can configure the publication of the OpenVPN server in several ways:

1. Select one or more external IP addresses for connection.
2. Choose one of the connection methods:
   - via external IP address (`ExternalIP`) — if there are nodes with public IPs;
   - using `LoadBalancer` — for all cloud providers and their deployment schemes that support LoadBalancer provisioning;
   - `Direct` — manually configure the traffic path from the cluster entry point to the OpenVPN pod.

Example for bare-metal clusters

Apply the following YAML file in your cluster:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    inlet: ExternalIP
    externalIP: 5.4.54.4 # External IP address.
```

Example for AWS and Google Cloud

Apply the following YAML file in your cluster:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    inlet: LoadBalancer
```

Example for a public IP address on an external Load Balancer

Apply the following YAML file in your cluster:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    externalHost: 5.4.54.4
    externalIP: 192.168.0.30 # Internal IP address that will receive traffic from the
external Load Balancer.
    inlet: ExternalIP
    nodeSelector:
      kubernetes.io/hostname: node
```

Authentication

You can configure user authentication to access OpenVPN. Alternatively, you can use the externalAuthentication parameter.
If both options are disabled, basic authentication with a generated password will be enabled.

To view the generated password, run the following command:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values openvpn -o json | jq '.openvpn.internal.auth.password'
```

To generate a new password, delete the Secret resource:

```
d8 k -n d8-openvpn delete secret/basic-auth
```

> **i** The `auth.password` parameter is no longer supported.

Available cluster resources after connecting to VPN

After connecting to the VPN, the following parameters are pushed to the user's computer:

- The `kube-dns` address is added to the client's DNS servers to allow direct access to Kubernetes services via FQDN;
- A route to the local network;
- A route to the cluster service network;
- A route to the pod network.

Auditing user connections

To monitor user connections, you can enable logging of VPN activity in JSON format. Traffic is grouped by the following fields: `src_ip`, `dst_ip`, `src_port`, `dst_port`, `ip_proto`.
Using log-shipper, you can collect and store container logs for later auditing.

Why automatic DNS configuration doesn't work on macOS and Linux with the OpenVPN client

Due to the architectural specifics of Linux and macOS, automatic DNS server configuration when using the official OpenVPN client is not supported.

These operating systems allow using external scripts that run on VPN connect and disconnect to configure DNS.

In the client configurations generated by the module, the following blocks are predefined and commented out:

```
# Uncomment the lines below for use with Linux
#script-security 2
# If you use resolved
#up /etc/openvpn/update-resolv-conf
#down /etc/openvpn/update-resolv-conf
# If you use systemd-resolved, first install the openvpn-systemd-resolved package
#up /etc/openvpn/update-systemd-resolved
#down /etc/openvpn/update-systemd-resolved
```

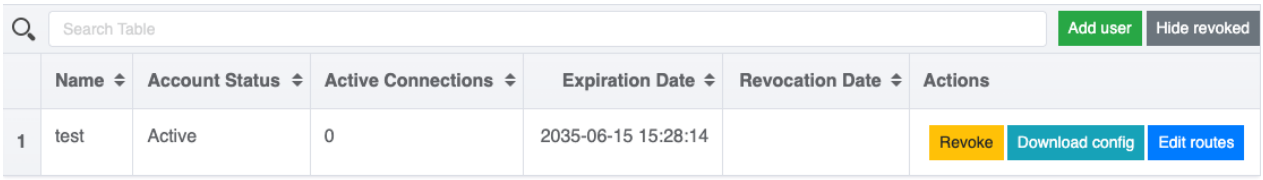
To activate these code blocks, you need to uncomment them (remove the leading `#`) and provide correct paths to the scripts.

You can either write your own scripts or use existing solutions provided by the official OpenVPN Community (for Linux).
For macOS, you can use a third-party script.

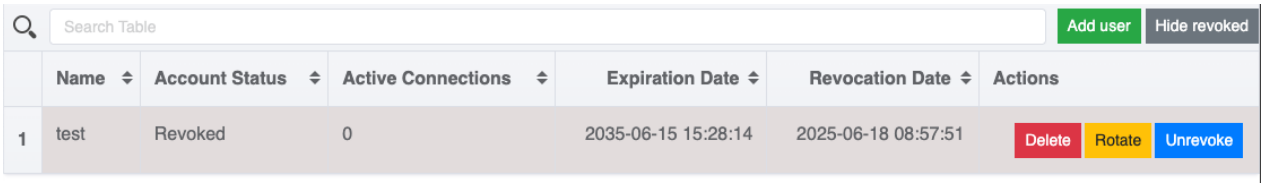
> **⚠** The scripts must have execution permissions.

How to revoke, rotate or delete a user certificate

All operations with client certificates are performed via the openvpn-admin web interface. Buttons for managing certificates are available to the right of each user's name:



To rotate (issue a new certificate) or delete a client, you must first revoke their current certificate:



Once revoked, the Renew (rotation) and Delete (removal) actions become available.

How to rotate a server certificate

The server certificate is rotated automatically several days before it expires.

If you need to rotate the certificate manually (e.g., due to certificate corruption or an unscheduled replacement), follow these steps:

1. Delete the secret `openvpn-pki-server` in the namespace `d8-openvpn`:

   ```
   d8 k -n d8-openvpn delete secrets openvpn-pki-server
   ```

2. Restart the OpenVPN pods to trigger the generation of a new certificate:

   ```
   d8 k -n d8-openvpn rollout restart sts openvpn
   ```

A new certificate will be generated automatically when the pods start.

How to rotate a root certificate (CA)

The root certificate (CA) and server certificate are rotated automatically 1 day before expiration. Automatic rotation of clients certificates is not provided. The root certificate (CA) is used to sign all certificates in OpenVPN — both server and client. Therefore, when replacing the CA, you must reissue all dependent certificates.

Steps to rotate the root certificate:

1. Revoke or delete all active client certificates using the openvpn-admin web interface. If you choose to revoke certificates, you can rotate them later (Renew) after the CA is replaced, without recreating the clients.
2. Delete secrets `openvpn-pki-ca` and `openvpn-pki-server` in the namespace `d8-openvpn`:

   ```
   d8 k -n d8-openvpn delete secrets openvpn-pki-ca openvpn-pki-server
   ```

3. Restart OpenVPN pods:

   ```
   d8 k -n d8-openvpn rollout restart sts openvpn
   ```

4. Rotate certificates of revoked users, or create new ones.
5. Delete all revoked certificate secrets:

   ```
   d8 k -n d8-openvpn delete secrets -l revokedForever=true
   ```

Additional resources

- Module `log-shipper` documentation

- Module `openvpn` parameters

Storage

Reliable data storage is one of the key tasks when deploying and operating Kubernetes clusters. In Deckhouse Kubernetes Platform, this is achieved through flexible support for both software-defined and external storage systems, as well as convenient automation and management tools.

In this section, you will learn:

- Which types of storage Deckhouse supports.
- How to set up both local (SDS) and external storage systems.
- Which tools Deckhouse provides to simplify storage configuration and operation.
- How to choose the optimal solution for your scenarios, based on requirements for reliability, performance, and scalability.

For more details on disk layout and recommended sizes, see the “Disk layout and size” guide.

Supported storage types

Deckhouse Kubernetes Platform offers a wide range of solutions, which can be divided into two main groups.

Software-defined storage

- Managing logical volumes on cluster nodes
- Local storage using Local Path Provisioner
- Local storage based on LVM (Logical Volume Manager)
- Replicated storage based on DRBD (Distributed Replicated Block Device)

External storage

- Distributed Ceph storage
- HPE data storage
- Huawei data storage
- NFS data storage
- S3-based object storage
- SCSI-based data storage
- TATLIN.UNIFIED (Yadro) unified storage

Key features

- All storage configurations are performed through Deckhouse and its modules, simplifying integration with all cluster components.
- Ready-to-use solutions for Ceph, NFS, S3, corporate storage systems, and other options, which make connecting storage to the cluster straightforward.
- Thanks to SDS support (including DRBD) and distributed solutions (Ceph, S3), you can scale up as your workload grows.
- Data replication and integration with robust storage systems help protect critical services and applications from data loss.

Storage configuration

Storage setup involves several steps depending on the selected storage type. The main configuration steps include:

- Enabling and configuring the corresponding modules.
- Creating Volume Groups.
- Preparing and creating StorageClass objects, followed by assigning and using them.

Each storage type may have its own specific requirements and configuration details, which are described in the corresponding guides.

Creating a StorageClass

To create StorageClass objects, you need to connect one or more storage backends that manage PersistentVolume resources. The created StorageClass objects can then be used to provision virtual disks and images. For more information on creating and using StorageClasses, refer to the appropriate documentation sections for each storage type.

Setting a default StorageClass

The default StorageClass is used when a PersistentVolumeClaim is created without explicitly specifying a storage class. This simplifies the process of creating and using storage by eliminating the need to define the class manually each time.

To set the default StorageClass, specify the desired class in the global configuration. Example command:

```
# Replace with the name of your StorageClass object.
DEFAULT_STORAGE_CLASS=replicated-storage-class
d8 k patch mc global --type='json' -p='[{"op": "replace", "path":
"/spec/settings/defaultClusterStorageClass", "value":
"'"$DEFAULT_STORAGE_CLASS"'"}]'
```

Additional resources

- Global parameters

Configuring volume snapshot creation

Deckhouse Kubernetes Platform supports volume snapshot creation for CSI drivers in a Kubernetes cluster.

Snapshots capture the state of a volume at a specific point in time and can be used for data recovery or volume cloning. The ability to create snapshots depends on the capabilities of the CSI driver in use.

Supported CSI drivers

The following CSI drivers support snapshot creation:

- OpenStack cloud provider
- VMware vSphere cloud provider
- Distributed storage based on Ceph
- Amazon Web Services cloud provider
- Microsoft Azure cloud provider
- Google Cloud Platform cloud provider
- Replicated storage based on DRBD
- NFS-based storage

Creating snapshots

Before creating snapshots, make sure that VolumeSnapshotClass resources are configured in the cluster. You can list available classes with the following command:

```
d8 k get volumesnapshotclasses.snapshot.storage.k8s.io
```

To create a snapshot for a volume, specify the required VolumeSnapshotClass in the manifest:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: example-snapshot
spec:
  volumeSnapshotClassName: <class-name>
  source:
    persistentVolumeClaimName: <pvc-name>
```

Restoring from a snapshot

To restore data from a snapshot, create a PVC that references the existing VolumeSnapshot:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: restored-pvc
spec:
  dataSource:
    name: example-snapshot
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
  storageClassName: <storage-class-name>
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

⚠ Not all CSI drivers support restoring volumes from snapshots. Ensure that your driver provides the required capabilities.

Additional resources

- Module `cloud-provider-aws` documentation
- Module `cloud-provider-azure` documentation
- Module `cloud-provider-gcp` documentation
- Module `cloud-provider-openstack` documentation
- Module `cloud-provider-vsphere` documentation

Local path provisioner storage

Deckhouse Kubernetes Platform provides the ability to configure local storage using Local Path Provisioner. This is a simple solution without support for snapshots or volume size limits, best suited for development, testing, and small clusters. It uses local disk space on Kubernetes nodes to create PersistentVolumes without relying on external storage systems.

How it works

For each LocalPathProvisioner resource, a corresponding `StorageClass` object is created. The list of nodes allowed to use the StorageClass is defined by the `nodeGroups` field and is used when scheduling pods.

When a pod requests a disk, the following occurs:

- A PersistentVolume of type `HostPath` is created;
- A directory is created on the appropriate node, with the `path` generated from the path parameter, the PV name, and the PVC name.

Example path:

```
/opt/local-path-provisioner/pvc-d9bd3878-f710-417b-a4b3-38811aa8aac1_d8-monitoring_prometheus-main-db-prometheus-main-0
```

Limitations

- It is not possible to set a size limit for created volumes.

Example LocalPathProvisioner resources

ReclaimPolicy: Retain (default)

```
apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
spec:
  nodeGroups:
  - system
  path: "/opt/local-path-provisioner"
```

ReclaimPolicy: Delete

```
apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
spec:
  nodeGroups:
  - system
  path: "/opt/local-path-provisioner"
  reclaimPolicy: "Delete"
```

Configuring Prometheus to use local storage

1. Apply the LocalPathProvisioner resource:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: LocalPathProvisioner
   metadata:
     name: localpath-system
   spec:
     nodeGroups:
     - system
     path: "/opt/local-path-provisioner"
   ```

2. Ensure that `spec.nodeGroups` matches the NodeGroup where Prometheus will be running.
3. Specify the name of the created StorageClass in the Prometheus configuration:

   ```
   longtermStorageClass: localpath-system
   storageClass: localpath-system
   ```

4. Wait for Prometheus pods to restart.

Additional resources

- Module `local-path-provisioner` custom resources

Node configuration for software-defined storage

Deckhouse Kubernetes Platform provides automatic management of logical volumes (Logical Volume Manager, LVM) on cluster nodes using custom Kubernetes resources. This functionality is provided by the `sds-node-configurator` module and includes:

- Detection of block devices on each node and creation of corresponding BlockDevice resources.

  > **i** Manual creation and modification of the BlockDevice resource is prohibited.

- Automatic discovery and management of LVMVolumeGroup resources with the label `storage.deckhouse.io/enabled=true` (including thin pools) on cluster nodes. When Volume Groups (VG) are detected on nodes without corresponding resources, the controller automatically creates the LVMVolumeGroup resource.

- Regular scanning of Physical Volumes (PV) that are part of managed Volume Groups (VG) on nodes. Upon expansion of the underlying block device, the controller performs `pvresize` on the corresponding physical volume and automatically increases the size of all logical volumes in this volume group.

  > **i** Reducing the size of the block device is not supported.

- Synchronization of the LVMVolumeGroup state with the actual state of the node: creation of new volume groups according to LVMVolumeGroup, expansion of existing ones when `desiredCapacity` changes, and deletion of the group when the resource is deleted. For detailed examples of operation, see the section Examples of working with LVMVolumeGroup.

Additional resources

- Module `sds-node-configurator` documentation

Usage

> **i** Functionality is guaranteed only when using stock kernels supplied with supported distributions. When using non-standard kernels or distributions, behavior may be unpredictable.

The controller operates with two types of Kubernetes custom resources:

- BlockDevice: A resource representing a block device.
- LVMVolumeGroup: A resource describing a Logical Volume Manager volume group.

Working with BlockDevice resources

Creating a BlockDevice resource

The controller periodically scans available block devices on each node and automatically creates a BlockDevice resource when it detects a device that matches the rules. As a result, a resource object with a unique name is created, containing detailed information about the device’s characteristics.

Criteria for device selection by the controller

- The device is not used as a DRBD volume (Distributed Replicated Block Device).
- The device is not a loop interface (virtual block device).
- The device is a logical volume.
- The file system is absent or has the label `LVM2_MEMBER` (indicating it belongs to an LVM device).
- The device does not contain partitions (no partition table).
- The device’s capacity exceeds 1 GiB.
- For virtual disks, a serial number is required.

The created BlockDevice resource serves as a data source for subsequent work with LVMVolumeGroup resources.

Updating a BlockDevice resource

When the state of a block device changes (for example, size change, metadata change, or the device disappears from the system), the controller automatically detects these changes and updates the fields of the BlockDevice resource, ensuring the information is up-to-date. User edits to the BlockDevice resource are prohibited and will be overwritten by the controller.

Deleting a BlockDevice resource

The controller deletes a BlockDevice resource only when the following conditions are met:

- The resource is in the `Consumable` state (available for consumption in the Logical Volume Manager).
- The block device is no longer available in the system (the device has been removed or disconnected).
- The block device is not part of a volume group with the label `storage.deckhouse.io/enabled=true` (such a volume group is managed by LVMVolumeGroup, and the BlockDevice is not deleted).

> **i** A BlockDevice resource is deleted without a help from a user. If the user manually deletes the resource, the controller will recreate it during the next scan if the device still meets the criteria.

Working with LVMVolumeGroup resources

The LVMVolumeGroup resource combines several BlockDevice resources into a single Logical Volume Manager (LVM) volume group on a node and reflects the current state of this group.

> **i** Currently, only the `Local` type (local volume groups) is supported.

Creating an LVMVolumeGroup resource

There are two scenarios for creating an LVMVolumeGroup resource:

- Automatic creation of the resource:
  - The controller scans the list of active LVM volume groups on each node.
  - If a discovered volume group has the label `storage.deckhouse.io/enabled=true` and there is no corresponding LVMVolumeGroup resource, the controller automatically creates the resource.
  - The controller fills out the `spec` section, except for the `thinPools` field. All other parameters (block device names, Volume Group name, etc.) are automatically pulled from the system state.
  - To manage thin pools, the user can manually add information about them to the `spec` section after the resource is created.
- Manual creation of the resource:
  - The user creates a YAML manifest, specifying the minimum set of fields:
    - `metadata.name`: Unique name for the LVMVolumeGroup resource.
    - `spec`: Name of the node where the volume group will exist.
  - After validation, the controller creates the volume group on the node and updates the custom resource with current information about the state of the created LVM Volume Group.

  Example of creating a local Volume Group without a thin pool:

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-0-on-node-0"
spec:
  type: Local
  local:
    nodeName: "node-0"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-07ad52cef2348996b72db262011f1b5f896bb68f
          - dev-e90e8915902bd6c371e59f89254c0fd644126□7
  actualVGNameOnTheNode: "vg-0"
```

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-0-on-node-0"
spec:
  type: Local
  local:
    nodeName: "node-0"
  blockDeviceSelector:
    matchLabels:
      kubernetes.io/hostname: node-0
  actualVGNameOnTheNode: "vg-0"
```

Example of creating a local Volume Group with a thin pool (250 GiB):

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-0-on-node-0"
spec:
  type: Local
  local:
    nodeName: "node-0"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-07ad52cef2348996b72db262011f1b5f896bb68f
          - dev-e90e8915902bd6c371e59f89254c0fd644126□7
  actualVGNameOnTheNode: "vg-0"
  thinPools:
    - name: thin-1
      size: 250Gi
```

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-0-on-node-0"
spec:
  type: Local
  local:
    nodeName: "node-0"
  blockDeviceSelector:
    matchLabels:
      kubernetes.io/hostname: node-0
  actualVGNameOnTheNode: "vg-0"
  thinPools:
    - name: thin-1
      size: 250Gi
```

The user can use any valid selectors (`matchLabels` or `matchExpressions`) to select BlockDevice resources:

- `matchLabels` allows selecting all devices that have the specified label (for example, `kubernetes.io/hostname=node-0`).
- `matchExpressions` allows for more flexible expressions, such as including listed device names.

The `spec.local.nodeName` field must match the name of the node where the Volume Group is created. Otherwise, the resource will not be started. All selected block devices must physically reside on the same node for a resource of type Local.

⚠ After applying the manifest, the controller will automatically create or update the LVM infrastructure on the node, including working with physical volumes (`pvcreate`), creating or modifying the volume group (`vgcreate`/`vgextend`), and setting up the thin pool (`lvcreate`).

Updating an LVMVolumeGroup resource

To change the configuration of the volume group (adding/removing block devices or changing thin pool parameters), it is sufficient to edit the `spec` section of the resource. The controller will validate the new parameters and adjust the state on the node (execute commands like `vgextend`, `lvcreate`, `lvresize`, etc.).

The `status` section of the resource automatically reflects the current information about the state of the Volume Group (list of physical volumes, thin pool data, free/used space, etc.).

⚠
- The controller does not change the `spec` automatically — it only reads the desired state and brings the actual state of the node into compliance. All changes to the desired state must be made by the user in `spec`.
- Avoid modifying `status` manually, as this section is managed by the controller.

Deleting an LVMVolumeGroup resource

The controller automatically deletes the resource if the Volume Group no longer exists on the node (for example, all block devices are disconnected or removed). For manual deletion of the Volume Group and its associated logical volumes, the user can use:

```
d8 k delete lvg <resource-name>
```

After that, the controller will perform a full removal of the physical and logical LVM objects (`lvremove`, `vgremove`, etc.).

⚠ If there are logical volumes (including thin pools) in the Volume Group, the user must first delete or move them (using commands `lvremove` or `pvmove` + `vgreduce`). Otherwise, the controller will not be able to delete the resource. To prevent accidental deletion of the resource, you can set the annotation `storage.deckhouse.io/deletion-protection` — while it is present, the controller will not initiate the deletion of the volume group.

Removing a BlockDevice from an LVMVolumeGroup

1. To exclude a specific BlockDevice from the Volume Group, adjust the `spec.blockDeviceSelector` selector of the LVMVolumeGroup resource or remove the corresponding label from the BlockDevice resource.
2. On the node, manually transfer data (if there are volumes in the Volume Group) using the following command:

   ```
   pvmove <source-device>
   ```
3. Remove the physical volume from the group:

   ```
   vgreduce <VG-name> <excluded-device>
   ```
4. Finally, remove the physical volume metadata:

   ```
   pvremove <excluded-device>
   ```

i If logical volumes remain during the removal process, delete them in advance using the `lvremove` command. To protect against unintentional deletion, you can use the annotation `storage.deckhouse.io/deletion-protection` — until it is removed, the LVMVolumeGroup resource will not be deleted.

Protecting against data leaks between volumes

When files are deleted, the operating system does not erase the data on the disk: the corresponding blocks are merely marked as “free”. If the same physical blocks are used when creating a new volume, they may contain residual data from the previous user.

For example:

- User #1 created a volume from StorageClass #1 on node #1 (regardless of the “Block” or “Filesystem” mode) and wrote files to it.
- User #1 deleted the files and the volume itself.
- The blocks that the volume occupied were marked as free but not overwritten.
- User #2 requested a new volume from StorageClass #1 on the same node #1 in block mode.
- There is a risk that some or all of the previously freed blocks will be allocated to user #2 again.
- As a result, user #2 may gain access to user #1’s data.

Thick volume

To ensure confidentiality, the `volumeCleanup` parameter is used in the `PersistentVolume` configuration. Available modes:

- Parameter not set: When the volume is deleted, no additional operations are performed, the blocks remain uninitialized, and the data may be accessible to the next user.
- `RandomFillSinglePass`: The volume is overwritten once with random data (it is recommended to avoid this on solid-state drives due to wear).
- `RandomFillThreePass`: The volume is overwritten three times with random data (maximum reliability but high load on the device).
- `Discard`: Using TRIM, all block devices are marked as free. Effective on most modern solid-state drives, but depends on the SSD controller’s support for DRAT/RZAT.

Using Discard mode

Most modern solid-state drives support the `discard` (TRIM) command, which marks freed blocks as “zero” and does not return old data on subsequent reads. Therefore, using the `Discard` option is the most reliable way to prevent data leaks on solid-state drives. However, the following nuances should be considered:

- The `discard` command initiates cell cleaning, but the actual zeroing is performed by the drive’s controller in the background. Until this operation is completed, some blocks may still contain old data.
- Not all solid-state drives clean specific physical blocks individually: many devices work at the page level and larger blocks, so individual logical blocks may remain uncleaned until the entire page is fully processed.
- To be sure that after `discard` the blocks are indeed read as zeros, the disk must support DRAT (Deterministic TRIM) — predictable behavior of the `discard` command and RZAT (Deterministic Read Zero after TRIM) — guaranteed reading of “zeros” after `discard`. If the device has not confirmed support for DRAT and RZAT, there is no guarantee that the freed blocks will never return the old data.
- Even with declared support for DRAT/RZAT, manufacturers may not fully comply with the specifications. In the absence of independent tests or certification of solid-state drives, it is recommended to refrain from using the `Discard` mode for critical tasks related to data security.

> ⚠ If the drive is not confirmed to support DRAT and RZAT (meaning, has not passed verification for deterministic cleaning and reading of zeroed blocks), the use of the `Discard` option is **not recommended**, as there remains a risk of recovering old data.

Thin volume

When a block of a thin volume is released, the `discard` command from the guest OS is passed directly to the storage device. If a mechanical hard drive or a solid-state drive without `discard` support is used, the freed blocks in the thin pool remain with the previous data until they are physically overwritten. However, clients are provided only with thin volumes, not access to the thin pool itself. When a block is next allocated to a thin-LV, LVM guaranteedly fills the corresponding block in the thin pool with zeros (zero-fill), thanks to the `thin_pool_zero=1` parameter. This ensures the absence of residual data between different thin volumes and prevents information leaks between clients.

Additional resources

- Module `sds-node-configurator` custom resources

Configuration scenarios

Functionality is guaranteed only when using stock kernels supplied with supported distributions. When using non-standard kernels or distributions, behavior may be unpredictable.

Cloning virtual machines

When creating virtual machines using cloning methods, replace the UUID of the volume groups (VG) by executing:

```
vgchange -u
```

This command will generate new UUIDs for all VGs on the virtual machine. If necessary, the command can be added to the `cloud-init` script.

Changing the UUID is only possible if there are no active logical volumes (LV) in the group. They can be deactivated as follows:

```
lvchange -an <VG_or_LV_NAME>
```

Where `<VG_or_LV_NAME>` is the name of the VG to deactivate all volumes in the group, or the name of the LV to deactivate a specific volume.

Methods and scenarios for configuring the disk subsystem of nodes

The disk subsystem of each node can be organized in two ways, depending on whether the disks installed in the server are identical.

- Storage with identical disks: All disks in the node are of the same type and size.
- Combined storage: The node has disks of different types (for example, SSD + HDD).

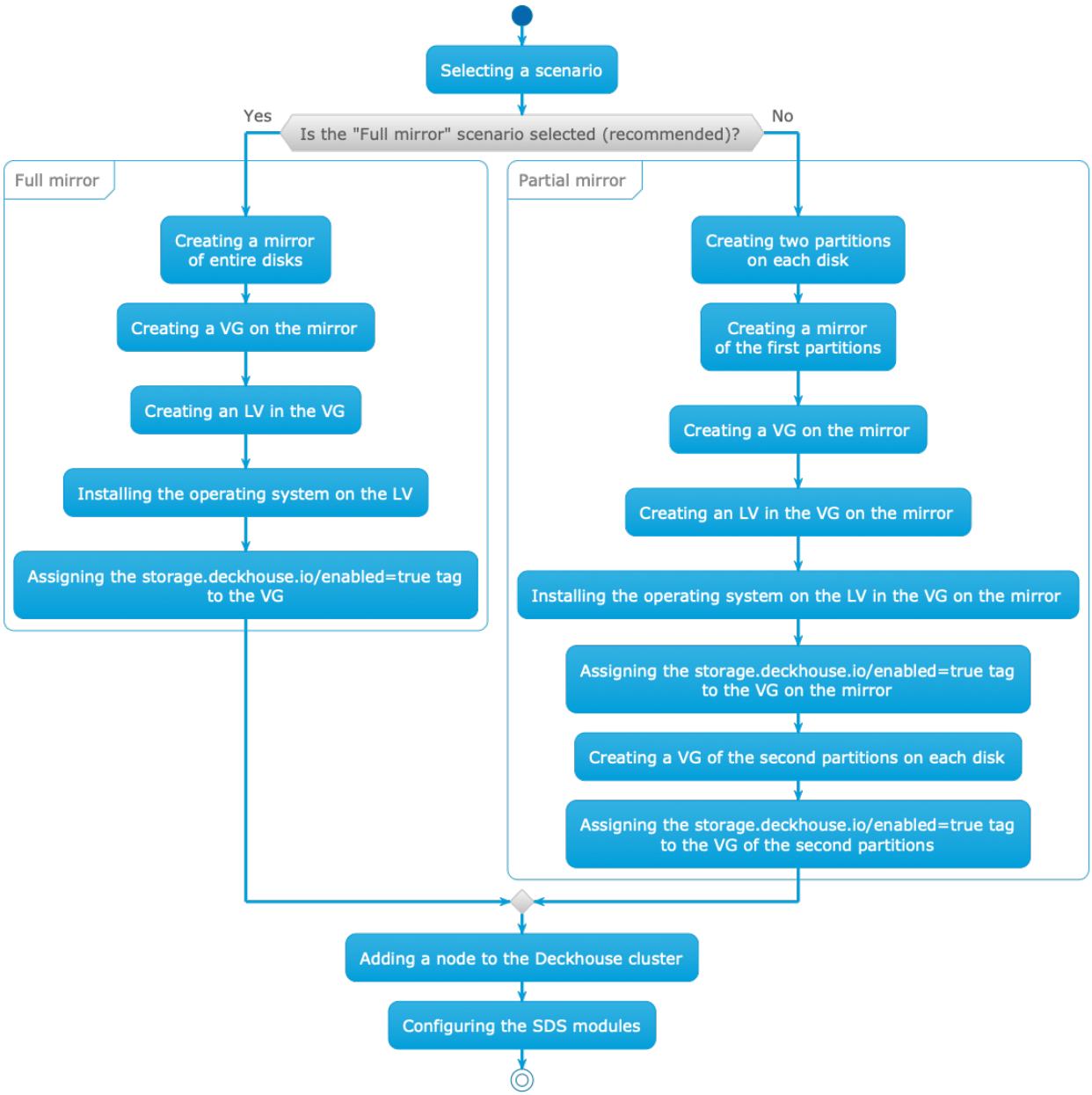
For each method of configuring the disk subsystem on nodes, there are two configuration scenarios:

- **Full mirror**: Recommended, reliable, and the simplest.
- **Partial mirror**: More flexible but requires caution.

The features, advantages, and disadvantages of the scenarios are presented in the table:

Configuration scenario	Implementation features	Advantages	Disadvantages
Full mirror	• Disks are not partitioned; a mirror is created from entire disks • Single VG for OS and data	• High reliability • Ease of setup and operation • Flexible resource allocation between SDS	• Excessive disk consumption with SDS replication
Partial mirror	• Disks are divided into 2 partitions • A mirror is created from the first partitions of each disk, on which a VG for the OS is created • A VG for data is created from the second partitions without mirroring	• Reliable storage • Maximum efficiency in space utilization	• Complex setup • Difficulties in reallocating space between safe and unsafe partitions

The differences in the configuration order of the disk subsystem depending on the chosen configuration scenario are illustrated in the diagram:



Storage with identical disks

Full mirror

We recommend using this configuration scenario as it is sufficiently reliable and easy to set up.

To configure a node according to this scenario, follow these steps:

1. Create a mirror from all disks (hardware or software). This mirror will be used simultaneously for the root system and for data.
2. Install the operating system:
   - Create a VG named `main` on the mirror.
   - Create an LV named `root` in the VG `main`.
   - Install the operating system on the LV `root`.
3. Set the tag `storage.deckhouse.io/enabled=true` for the VG `main` using the following command:

   ```
   vgchange main --addtag storage.deckhouse.io/enabled=true
   ```

4. Add the prepared node to the Deckhouse cluster.

   If the node matches the `nodeSelector` specified in `spec.nodeSelector` of the `sds-replicated-volume` or `sds-local-volume` modules, the `sds-node-configurator` agent will detect the VG

`main` and create a LVMVolumeGroup resource. This can be used in the `sds-local-volume` and `sds-replicated-volume` modules.

Example of configuring SDS modules (identical disks, “Full mirror”)

In this scenario, three nodes of the Deckhouse cluster are configured in “Full mirror” mode.
After automatic discovery, three CRD resources of type LVMVolumeGroup with automatically generated names will appear in the cluster.

To list the LVMVolumeGroup resources, execute the following command:

```
d8 k get lvmvolumegroups.storage.deckhouse.io
```

The result will be the following list:

```
NAME                                     THINPOOLS   CONFIGURATION APPLIED   PHASE
NODE       SIZE      ALLOCATED SIZE   VG     AGE
vg-08d3730c-9201-428d-966c-45795cba55a6   0/0         True                    Ready
worker-2   25596Mi   0                main   61s
vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d   0/0         True                    Ready
worker-0   25596Mi   0                main   4m17s
vg-c7863e12-c143-42bb-8e33-d578ce50d6c7   0/0         True                    Ready
worker-1   25596Mi   0                main   108s
```

Configuring the sds-local-volume module (identical disks, “Full mirror”)

To configure `sds-local-volume` in “Full Mirror” mode, create a LocalStorageClass resource and specify all discovered LVMVolumeGroup resources in it. This ensures that the VG with the label `main` is available on each node in the module:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: local-sc
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-08d3730c-9201-428d-966c-45795cba55a6
      - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
      - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

Configuring the sds-replicated-volume module (identical disks, “Full mirror”)

To configure the `sds-replicated-volume` module according to the “Full mirror” scenario, follow these steps:

1. Create a ReplicatedStoragePool resource and add all LVMVolumeGroup resources to it, so that the VG `main` is used on all nodes in the `sds-replicated-volume` module:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: data
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-08d3730c-9201-428d-966c-45795cba55a6
    - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
    - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
EOF
```

2. Create a ReplicatedStorageClass resource and specify the name of the previously created ReplicatedStoragePool resource in the `storagePool` field:

```
d8 k apply -f -<<EOF
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r1
spec:
  storagePool: data
  replication: None
  reclaimPolicy: Delete
  topology: Ignored # If this topology is specified, there should be no zones
in the cluster (nodes with labels topology.kubernetes.io/zone).
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r2
spec:
  storagePool: data
  replication: Availability
  reclaimPolicy: Delete
  topology: Ignored
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r3
spec:
  storagePool: data
  replication: ConsistencyAndAvailability
  reclaimPolicy: Delete
  topology: Ignored
EOF
```

Partial mirror

⚠

- Do not use partitions with the same `PARTUUID`.
- Changing the `PARTUUID` of a partition on which a VG is already created is not supported.
- It is recommended to use GPT for the partition table — in MBR, `PARTUUID` is pseudo-random and contains the partition number, and there is no support for `PARTLABEL`, which may be useful for identifying partitions in Deckhouse.

In this scenario, two partitions are used on each disk:

- A partition for the root system and storage of SDS data that is not replicated.
- A partition for SDS data that is replicated.

The first partition of each disk is used to create a mirror, and the second is used to create a separate VG without mirroring. This allows for the most efficient use of disk space.

To configure a node according to the “Partial mirror” scenario, follow these steps:

1. During the operating system installation:
   - Create two partitions on each disk.
   - Assemble a mirror from the first partitions on each disk.
   - Create a VG named `main-safe` on the mirror.
   - Create an LV named `root` in the VG `main-safe`.
   - Install the operating system on the LV `root`.
2. Set the tag `storage.deckhouse.io/enabled=true` for the VG `main-safe` using the following command:

   ```
   vgchange main-safe --addtag storage.deckhouse.io/enabled=true
   ```
3. Create a VG named `main-unsafe` from the second partitions of each disk.
4. Set the tag `storage.deckhouse.io/enabled=true` for the VG `main-unsafe` using the following command:

   ```
   vgchange main-unsafe --addtag storage.deckhouse.io/enabled=true
   ```
5. Add the prepared node to the Deckhouse cluster.

   If the node matches the `nodeSelector` specified in `spec.nodeSelector` of the `sds-replicated-volume` or `sds-local-volume` modules, the `sds-node-configurator` agent will run on this node, detect the VGs `main-safe` and `main-unsafe`, and add the corresponding LVMVolumeGroup resources to the Deckhouse cluster. These LVMVolumeGroup resources can then be used to create volumes in the `sds-replicated-volume` or `sds-local-volume` modules.

Example of configuring SDS modules (identical disks, “Partial mirror”)

In this example, it is assumed that you have configured three nodes according to the “Partial mirror” scenario. In the Deckhouse cluster, six LVMVolumeGroup resources with randomly generated names will appear. In the future, it will be possible to specify a name for the LVMVolumeGroup resources created during the automatic VG discovery process using the `LVM` tag with the desired resource name.

To list the LVMVolumeGroup resources, execute the following command:

```
d8 k get lvmvolumegroups.storage.deckhouse.io
```

The result will be the following list:

```
NAME                                   THINPOOLS   CONFIGURATION APPLIED   PHASE
NODE       SIZE      ALLOCATED SIZE   VG            AGE
vg-08d3730c-9201-428d-966c-45795cba55a6   0/0         True                    Ready
worker-2   25596Mi   0                main-safe     61s
vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d   0/0         True                    Ready
worker-0   25596Mi   0                main-safe     4m17s
vg-c7863e12-c143-42bb-8e33-d578ce50d6c7   0/0         True                    Ready
worker-1   25596Mi   0                main-safe     108s
vg-deccf08a-44d4-45f2-aea9-6232c0eeef91   0/0         True                    Ready
worker-2   25596Mi   0                main-unsafe   61s
vg-e0f00cab-03b3-49cf-a2f6-595628a2593c   0/0         True                    Ready
worker-0   25596Mi   0                main-unsafe   4m17s
vg-fe679d22-2bc7-409c-85a9-9f0ee29a6ca2   0/0         True                    Ready
worker-1   25596Mi   0                main-unsafe   108s
```

Configuring the sds-local-volume module (identical disks, “Partial mirror”)

To configure the `sds-local-volume` module according to the “Partial mirror” scenario, create a LocalStorageClass resource and add the LVMVolumeGroup resources to it so that only the VG `main-`

`safe` is used on all nodes in the `sds-local-volume` module:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: local-sc
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-08d3730c-9201-428d-966c-45795cba55a6
      - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
      - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

Configuring the sds-replicated-volume module (identical disks, “Partial mirror”)

To configure the `sds-replicated-volume` module according to the “Partial mirror” scenario, follow these steps:

1. Create a ReplicatedStoragePool resource named `data-safe` and add the LVMVolumeGroup resources to it so that only the VG `main-safe` is used on all nodes in the `sds-replicated-volume` module for ReplicatedStorageClass with the parameter `replication: None`:

   ```
   d8 k apply -f -<<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: ReplicatedStoragePool
   metadata:
     name: data-safe
   spec:
     type: LVM
     lvmVolumeGroups:
       - name: vg-08d3730c-9201-428d-966c-45795cba55a6
       - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
       - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
   EOF
   ```

2. Create a ReplicatedStoragePool resource named `data-unsafe` and add the LVMVolumeGroup resources to it so that only the VG `main-unsafe` is used on all nodes in the `sds-replicated-volume` module for ReplicatedStorageClass with the parameter `replication: Availability` or `replication: ConsistencyAndAvailability`:

   ```
   d8 k apply -f -<<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: ReplicatedStoragePool
   metadata:
     name: data-unsafe
   spec:
     type: LVM
     lvmVolumeGroups:
       - name: vg-deccf08a-44d4-45f2-aea9-6232c0eeef91
       - name: vg-e0f00cab-03b3-49cf-a2f6-595628a2593c
       - name: vg-fe679d22-2bc7-409c-85a9-9f0ee29a6ca2
   EOF
   ```

3. Create a ReplicatedStorageClass resource and specify the name of the previously created ReplicatedStoragePool resources in the `storagePool` field so that the VGs `main-safe` and `main-unsafe` are used on all nodes:

```
d8 k apply -f -<<EOF
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r1
spec:
  storagePool: data-safe # Note that due to replication: None, this resource
uses data-safe; therefore, data replication for persistent volumes (PV) created
with this StorageClass will not be performed.
  replication: None
  reclaimPolicy: Delete
  topology: Ignored # If this topology is specified, there should be no zones
in the cluster (nodes with labels topology.kubernetes.io/zone).
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r2
spec:
  storagePool: data-unsafe # Note that due to replication: Availability, this
resource uses data-unsafe; therefore, data replication for PVs created with
this StorageClass will be performed.
  replication: Availability
  reclaimPolicy: Delete
  topology: Ignored
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-r3
spec:
  storagePool: data-unsafe # Note that due to replication:
ConsistencyAndAvailability, this resource uses data-unsafe; therefore, data
replication for PVs created with this StorageClass will be performed.
  replication: ConsistencyAndAvailability
  reclaimPolicy: Delete
  topology: Ignored
EOF
```

Combined storage

Combined storage involves the simultaneous use of disks of different types on a node.

When combining disks of different types to create storage, we recommend creating a mirror from disks of one type and installing the operating system on it according to the “Full mirror” scenario, but not using it for SDS.

For SDS, use disks of other types (hereinafter referred to as additional disks) that differ from those used for the mirror under the operating system.

Recommendations for using additional disks depending on their type:

Disk type	Recommended use cases
NVMe SSD	Creating volumes that require high performance
SATA SSD	Creating volumes that do not require high performance
HDD	Creating volumes that do not require high performance

Additional disks can be configured according to either the “Full mirror” or “Partial mirror” scenarios.

Below, the process of configuring additional disks will be considered using the following types as

examples:

- NVMe SSD
- SATA SSD
- HDD

Configuring additional disks (Full mirror)

⚠ The following describes the procedure for configuring additional disks for the case of initial deployment and configuration of the cluster when connecting to nodes via SSH. If you already have a running cluster and are adding additional disks to its nodes, it is recommended to create and configure VGs using the LVMVolumeGroup resource instead of executing the commands below on the node.

To configure additional disks on a node according to the “Full mirror” scenario, follow these steps:

1. Assemble a mirror from all additional disks of a certain type entirely (hardware or software).
2. Create a VG named `<vg-name>` on the mirror.
3. Set the tag `storage.deckhouse.io/enabled=true` for the VG `<vg-name>` using the following command:

   ```
   vgchange <vg-name> --addtag storage.deckhouse.io/enabled=true
   ```

i In the example above, replace `<vg-name>` with an informative name depending on the type of additional disks.

Examples of VG names for additional disks of different types:

- `ssd-nvme`: For NVMe SSD disks.
- `ssd-sata`: For SATA SSD disks.
- `hdd`: For HDD disks.

Example of configuring SDS modules (combined storage, “Full mirror”)

In this example, it is assumed that you have configured three nodes according to the “Full Mirror” scenario. In the Deckhouse cluster, three LVMVolumeGroup resources with randomly generated names will appear. In the future, it will be possible to specify a name for the LVMVolumeGroup resources created during the automatic VG discovery process using the `LVM` tag with the desired resource name.

To list the LVMVolumeGroup resources, execute the following command:

```
d8 k get lvmvolumegroups.storage.deckhouse.io
```

The result will be a list like this:

```
NAME                                   THINPOOLS   CONFIGURATION APPLIED   PHASE
NODE       SIZE      ALLOCATED SIZE   VG          AGE
vg-08d3730c-9201-428d-966c-45795cba55a6   0/0         True                    Ready
worker-2   25596Mi   0                <vg-name>   61s
vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d   0/0         True                    Ready
worker-0   25596Mi   0                <vg-name>   4m17s
vg-c7863e12-c143-42bb-8e33-d578ce50d6c7   0/0         True                    Ready
worker-1   25596Mi   0                <vg-name>   108s
```

Where `<vg-name>` is the name assigned to the VG on the mirror in the previous step.

Configuring the sds-local-volume module (combined storage, “Full mirror”)

To configure the `sds-local-volume` module according to the “Full Mirror” scenario, create a LocalStorageClass resource and add all LVMVolumeGroup resources to it so that the VG `<vg-name>` is used on all nodes in the `sds-local-volume` module:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: <local-storage-class-name>
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-08d3730c-9201-428d-966c-45795cba55a6
      - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
      - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

i In the example above, replace `<local-storage-class-name>` with an informative name depending on the type of additional disks.

Examples of informative names for the LocalStorageClass resource for additional disks of different types:

- `local-sc-ssd-nvme`: For NVMe SSD disks.
- `local-sc-ssd-sata`: For SATA SSD disks.
- `local-sc-hdd`: For HDD disks.

Configuring the sds-replicated-volume module (combined storage, “Full mirror”)

To configure the `sds-replicated-volume` module according to the “Full Mirror” scenario, follow these steps:

1. Create a ReplicatedStoragePool resource and add all LVMVolumeGroup resources to it so that the VG `<vg-name>` is used on all nodes in the `sds-replicated-volume` module:

   ```
   d8 k apply -f -<<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: ReplicatedStoragePool
   metadata:
     name: <replicated-storage-pool-name>
   spec:
     type: LVM
     lvmVolumeGroups:
       - name: vg-08d3730c-9201-428d-966c-45795cba55a6
       - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
       - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
   EOF
   ```

   i In the example above, replace `<replicated-storage-pool-name>` with an informative name depending on the type of additional disks.

   Examples of informative names for the ReplicatedStoragePool resource for additional disks of different types:

   - `data-ssd-nvme`: For NVMe SSD disks.
   - `data-ssd-sata`: For SATA SSD disks.
   - `data-hdd`: For HDD disks.

2. Create a ReplicatedStorageClass resource and specify the name of the previously created ReplicatedStoragePool resource in the `storagePool` field:

```
d8 k apply -f -<<EOF
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r1
spec:
  storagePool: <replicated-storage-pool-name>
  replication: None
  reclaimPolicy: Delete
  topology: Ignored # If this topology is specified, there should be no zones
in the cluster (nodes with labels topology.kubernetes.io/zone).
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r2
spec:
  storagePool: <replicated-storage-pool-name>
  replication: Availability
  reclaimPolicy: Delete
  topology: Ignored
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r3
spec:
  storagePool: <replicated-storage-pool-name>
  replication: ConsistencyAndAvailability
  reclaimPolicy: Delete
  topology: Ignored
EOF
```

Configuring additional disks (Partial mirror)

⚠
- Do not use partitions with the same `PARTUUID`.
- Changing the `PARTUUID` of a partition on which a VG is already created is not supported.
- It is recommended to use GPT for the partition table — in MBR, `PARTUUID` is pseudo-random and contains the partition number, and there is no support for `PARTLABEL`, which may be useful for identifying partitions in Deckhouse.

⚠ The following describes the procedure for configuring additional disks for the case of initial deployment and configuration of the cluster when connecting to nodes via SSH. If you already have a running cluster and are adding additional disks to its nodes, it is recommended to create and configure VGs using the LVMVolumeGroup resource instead of executing the commands below on the node.

In this scenario, two partitions are used on each disk: one for storing SDS data that is not replicated and another for SDS data that is replicated. The first partition of each disk is used to create a mirror, and the second is used to create a separate VG without mirroring. This allows for the most efficient use of disk space.

To configure a node with additional disks according to the “Partial mirror” scenario, follow these steps:

1. Create two partitions on each additional disk.
2. Assemble a mirror from the first partitions on each disk.
3. Create a VG named `<vg-name>-safe` on the mirror.
4. Create a VG named `<vg-name>-unsafe` from the second partitions of each disk.
5. Set the tag `storage.deckhouse.io/enabled=true` for the VGs `<vg-name>-safe` and `<vg-name>-unsafe` using the following commands:

```
vgchange <vg-name>-safe --addtag storage.deckhouse.io/enabled=true
vgchange <vg-name>-unsafe --addtag storage.deckhouse.io/enabled=true
```

> **i** In the example above, replace `<vg-name>` with an informative prefix depending on the type of additional disks.
>
> Examples of informative prefixes `<vg-name>` for additional disks of different types:
>
> - `ssd-nvme`: For NVMe SSD disks.
> - `ssd-sata`: For SATA SSD disks.
> - `hdd`: For HDD disks.

Example of configuring SDS modules (Combined storage, “Partial mirror”)

In this example, it is assumed that you have configured three nodes according to the “Partial Mirror” scenario. In the Deckhouse cluster, six LVMVolumeGroup resources with randomly generated names will appear. In the future, it will be possible to specify a name for the LVMVolumeGroup resources created during the automatic VG discovery process using the `LVM` tag with the desired resource name.

To list the LVMVolumeGroup resources, execute the following command:

```
d8 k get lvmvolumegroups.storage.deckhouse.io
```

The result will be a list like this:

```
NAME                                   THINPOOLS   CONFIGURATION APPLIED   PHASE
NODE       SIZE      ALLOCATED SIZE   VG                AGE
vg-08d3730c-9201-428d-966c-45795cba55a6   0/0         True                    Ready
worker-2   25596Mi   0                <vg-name>-safe    61s
vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d   0/0         True                    Ready
worker-0   25596Mi   0                <vg-name>-safe    4m17s
vg-c7863e12-c143-42bb-8e33-d578ce50d6c7   0/0         True                    Ready
worker-1   25596Mi   0                <vg-name>-safe    108s
vg-deccf08a-44d4-45f2-aea9-6232c0eeef91   0/0         True                    Ready
worker-2   25596Mi   0                <vg-name>-unsafe  61s
vg-e0f00cab-03b3-49cf-a2f6-595628a2593c   0/0         True                    Ready
worker-0   25596Mi   0                <vg-name>-unsafe  4m17s
vg-fe679d22-2bc7-409c-85a9-9f0ee29a6ca2   0/0         True                    Ready
worker-1   25596Mi   0                <vg-name>-unsafe  108s
```

Where `<vg-name>` is the prefix of the name assigned to the VGs created in the previous step.

Configuring the sds-local-volume module (combined storage, “Partial mirror”)

To configure the `sds-local-volume` module according to the “Partial Mirror” scenario, create a LocalStorageClass resource and add the LVMVolumeGroup resources to it so that only the VG `<vg-name>-safe` is used on all nodes in the `sds-local-volume` module:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: <local-storage-class-name>
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-08d3730c-9201-428d-966c-45795cba55a6
      - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
      - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

i In the example above, replace `<local-storage-class-name>` with an informative name depending on the type of additional disks.

Examples of informative names for the LocalStorageClass resource for additional disks of different types:

- `local-sc-ssd-nvme`: For NVMe SSD disks.
- `local-sc-ssd-sata`: For SATA SSD disks.
- `local-sc-hdd`: For HDD disks.

Configuring the sds-replicated-volume module (combined storage, “Partial mirror”)

To configure the `sds-replicated-volume` module according to the “Partial mirror” scenario, follow these steps:

1. Create a ReplicatedStoragePool resource named `data-<vg-name>-safe` and add the LVMVolumeGroup resources to it so that only the VG `<vg-name>-safe` is used on all nodes in the `sds-replicated-volume` module for ReplicatedStorageClass with the parameter `replication: None`:

   ```
   d8 k apply -f -<<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: ReplicatedStoragePool
   metadata:
     name: data-<vg-name>-safe
   spec:
     type: LVM
     lvmVolumeGroups:
       - name: vg-08d3730c-9201-428d-966c-45795cba55a6
       - name: vg-b59ff9e1-6ef2-4761-b5d2-6172926d4f4d
       - name: vg-c7863e12-c143-42bb-8e33-d578ce50d6c7
   EOF
   ```

   i In the example above, replace `data-<vg-name>-safe` with an informative name depending on the type of additional disks.

   Examples of informative names for the ReplicatedStoragePool resource for additional disks of different types:

   - `data-ssd-nvme-safe`: For NVMe SSD disks.
   - `data-ssd-sata-safe`: For SATA SSD disks.
   - `data-hdd-safe`: For HDD disks.

2. Create a ReplicatedStoragePool resource named `data-<vg-name>-unsafe` and add the LVMVolumeGroup resources to it so that only the VG `<vg-name>-unsafe` is used on all nodes in the `sds-replicated-volume` module for ReplicatedStorageClass with the parameter `replication: Availability` or `replication: ConsistencyAndAvailability`:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: data-<vg-name>-unsafe
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-deccf08a-44d4-45f2-aea9-6232c0eeef91
    - name: vg-e0f00cab-03b3-49cf-a2f6-595628a2593c
    - name: vg-fe679d22-2bc7-409c-85a9-9f0ee29a6ca2
EOF
```

> **i** In the example above, replace `data-<vg-name>-unsafe` with an informative name depending on the type of additional disks.
>
> Examples of informative names for the ReplicatedStoragePool resource for additional disks of different types:
>
> - `data-ssd-nvme-unsafe`: For NVMe SSD disks.
> - `data-ssd-sata-unsafe`: For SATA SSD disks.
> - `data-hdd-unsafe`: For HDD disks.

3. Create a ReplicatedStorageClass resource and specify the name of the previously created ReplicatedStoragePool resources in the `storagePool` field so that the VGs `<vg-name>-safe` and `<vg-name>-unsafe` are used on all nodes:

```
d8 k apply -f -<<EOF
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r1
spec:
  storagePool: data-<vg-name>-safe # Note that due to replication: None, this
resource uses data-<vg-name>-safe; therefore, data replication for PVs created
with this StorageClass will not be performed.
  replication: None
  reclaimPolicy: Delete
  topology: Ignored # If this topology is specified, there should be no zones
in the cluster (nodes with labels topology.kubernetes.io/zone).
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r2
spec:
  storagePool: data-<vg-name>-unsafe # Note that due to replication:
Availability, this resource uses data-<vg-name>-unsafe; therefore, data
replication for PVs created with this StorageClass will be performed.
  replication: Availability
  reclaimPolicy: Delete
  topology: Ignored
---
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-sc-ssd-nvme-r3
spec:
  storagePool: data-<vg-name>-unsafe # Note that due to replication:
ConsistencyAndAvailability, this resource uses data-<vg-name>-unsafe;
therefore, data replication for PVs created with this StorageClass will be
performed.
  replication: ConsistencyAndAvailability
  reclaimPolicy: Delete
  topology: Ignored
EOF
```

i

In the example above, replace `data-<vg-name>-unsafe` with an informative name depending on the type of additional disks.

Examples of informative names for the ReplicatedStoragePool resource for additional disks of different types:

- `data-ssd-nvme-unsafe`: For NVMe SSD disks.
- `data-ssd-sata-unsafe`: For SATA SSD disks.
- `data-hdd-unsafe`: For HDD disks.

In a similar way, replace `data-<vg-name>-safe`.

Examples of informative names for the ReplicatedStoragePool resource for additional disks of different types:

- `data-ssd-nvme-safe`: For NVMe SSD disks.
- `data-ssd-sata-safe`: For SATA SSD disks.
- `data-hdd-safe`: For HDD disks.

Additional resources

- Module `sds-local-volume` documentation
- Module `sds-node-configurator` documentation

- Module `sds-replicated-volume` documentation

FAQ

> **i** Functionality is guaranteed only when using stock kernels supplied with supported distributions. When using non-standard kernels or distributions, behavior may be unpredictable.

Reasons why BlockDevice resources aren’t created in the cluster

Most often, BlockDevice resources are not created because the available devices do not pass the controller’s filters. Ensure that the devices meet the selection criteria.

Reasons why LVMVolumeGroup resources aren’t created in the cluster

- Absence of BlockDevice: The controller will not create an LVMVolumeGroup if there are no BlockDevice resources in the cluster specified in its specification.
- Absence of a tag: If BlockDevice resources are present but the LVMVolumeGroup is missing, check that the corresponding LVM group on the node has the tag `storage.deckhouse.io/enabled=true`.

Reasons why after deleting the LVMVolumeGroup, the resource and Volume Group remain

This situation is possible in two cases:

1. There are logical volumes in the Volume Group — the controller is not responsible for deleting logical volumes on the node, so if there are any logical volumes in the Volume Group created via the resource, they must be manually deleted. After that, both the LVMVolumeGroup resource and the Volume Group itself (including physical volumes) will be automatically deleted.
2. The resource has the annotation `storage.deckhouse.io/deletion-protection` — this annotation protects the resource from deletion, along with the associated Volume Group. Remove the annotation with the command:

   ```
   d8 k annotate lvg <resource-name> storage.deckhouse.io/deletion-protection-
   ```

   After that, the LVMVolumeGroup resource and the corresponding Volume Group will be automatically deleted.

Reasons for failed attempts to create a Volume Group using the LVMVolumeGroup resource

Most likely, the resource did not pass validation by the controller (unlike the Kubernetes schema). The reason can be found in the `status.message` field of the LVMVolumeGroup resource or in the controller logs. Check that the specified BlockDevice resources meet the conditions:

- The `consumable` field is set to `true`.
- For `spec.type: Local`, all BlockDevice resources belong to the same node.
- Current names of BlockDevice resources are used.

Behavior of the LVMVolumeGroup resource when one of the devices in the Volume Group is disabled

The LVMVolumeGroup resource remains as long as the corresponding LVM group exists. When a device becomes unavailable, the group enters an error state, which is reflected in the `status` field.

After the device is restored, the group returns to the `Healthy` status, and the resource status is automatically updated.

Transferring management of an existing Volume Group on the node to the controller

Add the tag `storage.deckhouse.io/enabled=true` to the desired Volume Group:

```
vgchange myvg-0 --addtag storage.deckhouse.io/enabled=true
```

The controller will create the corresponding LVMVolumeGroup resource and take over management of the group.

Disabling tracking of the LVM Volume Group by the controller

Remove the tag `storage.deckhouse.io/enabled=true`:

```
vgchange myvg-0 --deltag storage.deckhouse.io/enabled=true
```

The controller will stop tracking and delete the associated LVMVolumeGroup resource.

Reasons for automatic setting of the tag storage.deckhouse.io/enabled=true on the Volume Group

The controller adds the tag when creating the Volume Group through the LVMVolumeGroup resource.

When migrating from the `linstor` module to `sds-node-configurator` and `sds-replicated-volume`, all `linstor-*` tags are replaced with `storage.deckhouse.io/enabled=true` to transfer management to the new logic.

Using the LVMVolumeGroupSet resource to create LVMVolumeGroup

The LVMVolumeGroupSet resource allows template-based creation of LVMVolumeGroup on nodes. Currently, the PerNode strategy is supported — one resource per node that matches the selector.

Example specification of LVMVolumeGroupSet:

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroupSet
metadata:
  name: my-lvm-volume-group-set
  labels:
    my-label: my-value
spec:
  strategy: PerNode
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker: ""
  lvmVolumeGroupTemplate:
    metadata:
      labels:
        my-label-for-lvg: my-value-for-lvg
    spec:
      type: Local
      blockDeviceSelector:
        matchLabels:
          status.blockdevice.storage.deckhouse.io/model: <model>
      actualVGNameOnTheNode: <VG-name-on-the-node>
```

Labels added by the controller to BlockDevice resources

- `status.blockdevice.storage.deckhouse.io/type`: LVM type.
- `status.blockdevice.storage.deckhouse.io/fstype`: File system type.
- `status.blockdevice.storage.deckhouse.io/pvuuid`: Physical volume (PV) UUID.
- `status.blockdevice.storage.deckhouse.io/vguuid`: Volume group (VG) UUID.
- `status.blockdevice.storage.deckhouse.io/partuuid`: Partition UUID.
- `status.blockdevice.storage.deckhouse.io/lvmvolumegroupname`: Name of the LVMVolumeGroup resource.
- `status.blockdevice.storage.deckhouse.io/actualvgnameonthenode`: Name of the LVM Volume Group on the node.
- `status.blockdevice.storage.deckhouse.io/wwn`: WWN (World Wide Name) of the device.

- `status.blockdevice.storage.deckhouse.io/serial`: Serial number of the device.
- `status.blockdevice.storage.deckhouse.io/size`: Size of the device.
- `status.blockdevice.storage.deckhouse.io/model`: Model of the device.
- `status.blockdevice.storage.deckhouse.io/rota`: Rotational device flag.
- `status.blockdevice.storage.deckhouse.io/hotplug`: Hotplug capability.
- `status.blockdevice.storage.deckhouse.io/machineid`: Identifier of the machine where the device is installed.

Additional resources

- Module `sds-node-configurator` documentation
- Module `sds-replicated-volume` documentation

Setting up local storage based on LVM

Local storage reduces network latency and provides higher performance compared to remote storage accessed over a network. This approach is particularly effective in test environments and EDGE clusters. This functionality is provided by the `sds-local-volume` module.

Configuring local storage

To ensure the correct operation of the `sds-local-volume` module, follow these steps:

1. Configure LVMVolumeGroup. Before creating a StorageClass, you must create an LVMVolumeGroup resource for the `sds-node-configurator` module on the cluster nodes.
2. Enable the `sds-node-configurator` module. Ensure that the module is enabled **before** enabling the `sds-local-volume` module.
3. Create the corresponding StorageClasses. Creating a StorageClass for the CSI driver `local.csi.storage.deckhouse.io` by a user is **prohibited**.

The module supports two operating modes: LVM and LVMThin.

Quick start

All commands are executed on a machine with access to the Kubernetes API and administrator privileges.

Enabling modules

Enabling the `sds-node-configurator` module:

1. Create a ModuleConfig resource to enable the module:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-node-configurator
spec:
  enabled: true
  version: 1
EOF
```

2. Wait for the module to reach the `Ready` state. At this stage, it is not necessary to check the pods in the `d8-sds-node-configurator` namespace.

```
d8 k get modules sds-node-configurator -W
```

Enabling the `sds-local-volume` module:

1. Activate the `sds-local-volume` module. The example below starts the module with default settings, which will create service pods for the `sds-local-volume` component on all cluster nodes:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-local-volume
spec:
  enabled: true
  version: 1
EOF
```

2. Wait for the module to reach the `Ready` state.

   ```
   d8 k get modules sds-local-volume -w
   ```

3. Ensure that all pods in the `d8-sds-local-volume` and `d8-sds-node-configurator` namespaces are in the `Running` or `Completed` state and are running on all nodes where LVM resources are planned to be used.

   ```
   d8 k -n d8-sds-local-volume get pod -owide -w
   d8 k -n d8-sds-node-configurator get pod -o wide -w
   ```

Preparing nodes for storage creation

For storage to function correctly on nodes, the `sds-local-volume-csi-node` pods must be running on the selected nodes.

By default, these pods are launched on all cluster nodes. You can verify their presence using the command:

```
d8 k -n d8-sds-local-volume get pod -owide
```

The placement of `sds-local-volume-csi-node` pods is managed by specific labels (`nodeSelector`). These labels are set in the `spec.settings.dataNodes.nodeSelector` parameter of the module.

Configuring storage on nodes

To configure storage on nodes, you need to create LVM volume groups using LVMVolumeGroup resources. This example creates a Thick storage.

⚠ Before creating an LVMVolumeGroup resource, ensure that the `sds-local-volume-csi-node` pod is running on the respective node. This can be checked with the command:

```
d8 k -n d8-sds-local-volume get pod -owide
```

1. Retrieve all BlockDevice resources available in your cluster:

   ```
   d8 k get bd
   ```

   Example output:

```
NAME                                             NODE       CONSUMABLE   SIZE
PATH
dev-ef4fb06b63d2c05fb6ee83008b55e486aa1161aa   worker-0   false
976762584Ki    /dev/nvme1n1
dev-0cfc0d07f353598e329d34f3821bed992c1ffbcd   worker-0   false
894006140416   /dev/nvme0n1p6
dev-7e4df1ddf2a1b05a79f9481cdf56d29891a9f9d0   worker-1   false
976762584Ki    /dev/nvme1n1
dev-b103062f879a2349a9c5f054e0366594568de68d   worker-1   false
894006140416   /dev/nvme0n1p6
dev-53d904f18b912187ac82de29af06a34d9ae23199   worker-2   false
976762584Ki    /dev/nvme1n1
dev-6c5abbd549100834c6b1668c8f89fb97872ee2b1   worker-2   false
894006140416   /dev/nvme0n1p6
```

2. Create an LVMVolumeGroup resource for the `worker-0` node:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-1-on-worker-0" # The name can be any fully qualified resource name
in Kubernetes. This LVMVolumeGroup resource name will be used to create
LocalStorageClass in the future.
spec:
  type: Local
  local:
    nodeName: "worker-0"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-ef4fb06b63d2c05fb6ee83008b55e486aa1161aa
          - dev-0cfc0d07f353598e329d34f3821bed992c1ffbcd
  actualVGNameOnTheNode: "vg-1" # The name of the LVM VG that will be created
from the specified block devices on the node.
EOF
```

3. Wait for the created LVMVolumeGroup resource to transition to the `Ready` state:

```
d8 k get lvg vg-1-on-worker-0 -W
```

   If the resource has transitioned to the `Ready` state, it means that an LVM VG named `vg-1` has been created on the `worker-0` node from the block devices `/dev/nvme1n1` and `/dev/nvme0n1p6`.

4. Create an LVMVolumeGroup resource for the `worker-1` node:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-1-on-worker-1"
spec:
  type: Local
  local:
    nodeName: "worker-1"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          - dev-7e4df1ddf2a1b05a79f9481cdf56d29891a9f9d0
          - dev-b103062f879a2349a9c5f054e0366594568de68d
  actualVGNameOnTheNode: "vg-1"
EOF
```

5. Wait for the created LVMVolumeGroup resource to transition to the `Ready` state:

   ```
   d8 k get lvg vg-1-on-worker-1 -w
   ```

   If the resource has transitioned to the `Ready` state, it means that an LVM VG named `vg-1` has been created on the `worker-1` node from the block devices `/dev/nvme1n1` and `/dev/nvme0n1p6`.

6. Create an LVMVolumeGroup resource for the `worker-2` node:

   ```
   d8 k apply -f - <<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: LVMVolumeGroup
   metadata:
     name: "vg-1-on-worker-2"
   spec:
     type: Local
     local:
       nodeName: "worker-2"
     blockDeviceSelector:
       matchExpressions:
         - key: kubernetes.io/metadata.name
           operator: In
           values:
             - dev-53d904f18b912187ac82de29af06a34d9ae23199
             - dev-6c5abbd549100834c6b1668c8f89fb97872ee2b1
     actualVGNameOnTheNode: "vg-1"
   EOF
   ```

7. Wait for the created LVMVolumeGroup resource to transition to the `Ready` state:

   ```
   d8 k get lvg vg-1-on-worker-2 -w
   ```

   If the resource has transitioned to the `Ready` state, it means that an LVM VG named `vg-1` has been created on the `worker-2` node from the block devices `/dev/nvme1n1` and `/dev/nvme0n1p6`.

8. Create a LocalStorageClass resource:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: local-storage-class
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-1-on-worker-0
      - name: vg-1-on-worker-1
      - name: vg-1-on-worker-2
    type: Thick
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

For a thin volume:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LocalStorageClass
metadata:
  name: local-storage-class
spec:
  lvm:
    lvmVolumeGroups:
      - name: vg-1-on-worker-0
      - name: vg-1-on-worker-1
      - name: vg-1-on-worker-2
        thin:
          poolName: thin-1
    type: Thin
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

Important. In a LocalStorageClass with `type: Thick`, you cannot use an LVMVolumeGroup that contains at least one thin pool.

9. Wait for the created LocalStorageClass resource to transition to the `Created` state:

```
d8 k get lsc local-storage-class -w
```

10. Verify that the corresponding StorageClass has been created:

```
d8 k get sc local-storage-class
```

If a StorageClass named `local-storage-class` appears, the configuration of the `sds-local-volume` module is complete. Users can now create PVCs by specifying the StorageClass named `local-storage-class`.

Selecting a volume cleanup method after PV deletion

When files are deleted, the operating system does not physically erase the content but only marks the corresponding blocks as “free.” If a new volume receives physical blocks previously used by another volume, those blocks may still contain the previous user’s data.

This is possible, for example, in the following scenario:

- User #1 places files in a volume requested from StorageClass 1 on node 1 (in either “Block” or “Filesystem” mode).
- User #1 deletes the files and the volume.

- The physical blocks that were occupied become “free” but are not wiped.
- User #2 requests a new volume from StorageClass 1 on node 1 in “Block” mode.
- There is a risk that some or all of the blocks previously used by User #1 will be reallocated to User #2.
- In this case, User #2 may be able to recover User #1’s data.

Thick volumes

To prevent data leaks through thick volumes, the `volumeCleanup` parameter is provided. It allows you to select the volume cleanup method before deleting a PV.

Possible values:

- Parameter not set: No additional actions are performed when deleting the volume. Data may remain accessible to the next user.
- `RandomFillSinglePass`: The volume is overwritten with random data once before deletion. This option is not recommended for solid-state drives, as it reduces the drive’s lifespan.
- `RandomFillThreePass`: The volume is overwritten with random data three times before deletion. This option is not recommended for solid-state drives, as it reduces the drive’s lifespan.
- `Discard`: All volume blocks are marked as free using the `discard` system call before deletion. This option is only meaningful for solid-state drives.

  Most modern solid-state drives guarantee that a block marked with `discard` will not return previous data when read. This makes the `Discard` option the most effective way to prevent data leaks when using solid-state drives.

  However, erasing a cell is a relatively slow operation, so it is performed by the device in the background. Additionally, many drives cannot erase individual cells but only groups (pages). As a result, not all drives guarantee the immediate unavailability of freed data. Furthermore, not all drives that claim to guarantee this keep their promise. If a device does not guarantee Deterministic TRIM (DRAT), Deterministic Read Zero after TRIM (RZAT), and is not verified, it is not recommended for use.

Thin volumes

When a block in a thin volume is freed via `discard` by the guest operating system, this command is forwarded to the device. If a hard disk is used or if the solid-state drive does not support `discard`, the data may remain in the thin pool until the block is reused. However, users are only granted access to thin volumes, not the thin pool itself. They can only obtain a volume from the pool, and for thin volumes, the thin pool block is zeroed upon reuse, preventing data leaks between clients. This is ensured by the `thin_pool_zero=1` setting in LVM.

System requirements and recommendations

- Use stock kernels provided with supported distributions.
- Do not use another SDS (Software Defined Storage) to provide disks for SDS Deckhouse.

Additional resources

- Module `sds-local-volume` documentation
- Module `sds-node-configurator` documentation

Managing local storage based on LVM

Selecting specific nodes for module usage

To restrict the module’s usage to specific cluster nodes, you need to set labels in the `nodeSelector` field in the module settings.

To display and edit the module settings, execute the command:

```
d8 k edit mc sds-local-volume
```

Configuration example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-local-volume
spec:
  enabled: true
  settings:
    dataNodes:
      nodeSelector:
        my-custom-label-key: my-custom-label-value
status:
  message: ""
  version: "1"
```

To view the current labels in the `nodeSelector` field, use the following command:

```
d8 k get mc sds-local-volume -o=jsonpath={.spec.settings.dataNodes.nodeSelector}
```

Example output:

```
nodeSelector:
  my-custom-label-key: my-custom-label-value
```

The module selects only those nodes that have all the labels specified in the `nodeSelector`. By modifying this field, you can control the list of nodes on which the module will run.

> **i** You can specify multiple labels in the `nodeSelector`. However, for the module to work correctly, all these labels must be present on each node where you intend to run `sds-local-volume-csi-node`.

After configuring the labels, ensure that the `sds-local-volume-csi-node` Pods are running on the target nodes. You can check their presence with the command:

```
d8 k -n d8-sds-local-volume get pod -owide
```

Verifying PVC creation on the selected node

Make sure that the `sds-local-volume-csi-node` Pod is running on the selected node. To do this, run the command:

```
d8 k -n d8-sds-local-volume get po -owide
```

If the Pod is absent, verify that all the labels specified in the module settings in the nodeSelector field are present on the node. For available solutions when Pods are missing from the target node, refer to this section.

Removing a node from the module management

To remove a node from module management, you need to delete the labels set in the `nodeSelector` field in the `sds-local-volume` module settings.

To check the current labels, execute the command:

```
d8 k get mc sds-local-volume -o=jsonpath={.spec.settings.dataNodes.nodeSelector}
```

Example output:

```
nodeSelector:
  my-custom-label-key: my-custom-label-value
```

Remove the specified labels from the nodes with the command:

```
d8 k label node %node-name% %label-from-selector%-
```

⚠ After the label key, you must specify a minus sign to remove it.

After this, the `sds-local-volume-csi-node` Pod should be removed from the node. Check its status with the command:

```
d8 k -n d8-sds-local-volume get po -owide
```

If the Pod remains after removing the label, ensure that the labels from the `d8-sds-local-volume-controller-config` config are actually removed. You can verify this using:

```
d8 k get node %node-name% --show-labels
```

If the labels are absent, check that the node does not have any LVMVolumeGroup resources being used by LocalStorageClass resources. More information on this check can be found in this section.

⚠ Note that for LVMVolumeGroup and LocalStorageClass resources that are the reason why the node cannot be removed from module management, the label `storage.deckhouse.io/sds-local-volume-candidate-for-eviction` will be assigned.

On the node itself, the label `storage.deckhouse.io/sds-local-volume-need-manual-eviction` will be present.

Verifying dependent LVMVolumeGroup resources on the node

To check the dependent resources, follow these steps:

1. Display the available LocalStorageClass resources:

   ```
   d8 k get lsc
   ```

2. Check each of them for the list of used LVMVolumeGroup resources.

   If you want to list all LocalStorageClass resources at once, run the command:

   ```
   d8 k get lsc -oyaml
   ```

   Example LocalStorageClass resource:

   ```
   apiVersion: v1
   items:
   - apiVersion: storage.deckhouse.io/v1alpha1
     kind: LocalStorageClass
     metadata:
       finalizers:
       - localstorageclass.storage.deckhouse.io
       name: test-sc
     spec:
       lvm:
         lvmVolumeGroups:
         - name: test-vg
         type: Thick
       reclaimPolicy: Delete
       volumeBindingMode: WaitForFirstConsumer
     status:
       phase: Created
   kind: List
   ```

Pay attention to the `spec.lvm.lvmVolumeGroups` field. It specifies the used LVMVolumeGroup resources.

3. Display the list of existing LVMVolumeGroup resources.

   ```
   d8 k get lvg
   ```

   Example LVMVolumeGroup output:

   ```
   NAME              HEALTH        NODE            SIZE       ALLOCATED SIZE   VG        AGE
   lvg-on-worker-0   Operational   node-worker-0   40956Mi    0                test-vg   15d
   lvg-on-worker-1   Operational   node-worker-1   61436Mi    0                test-vg   15d
   lvg-on-worker-2   Operational   node-worker-2   122876Mi   0                test-vg   15d
   lvg-on-worker-3   Operational   node-worker-3   307196Mi   0                test-vg   15d
   lvg-on-worker-4   Operational   node-worker-4   307196Mi   0                test-vg   15d
   lvg-on-worker-5   Operational   node-worker-5   204796Mi   0                test-vg   15d
   ```

4. Ensure that the node you intend to remove from the module's control does not have any LVMVolumeGroup resources used in LocalStorageClass resources. To avoid unintentionally losing control over volumes already created using the module, the user needs to manually delete dependent resources by performing necessary operations on the volume.

Remaining sds-local-volume-csi-node Pod after removing labels

If after removing the labels from the node the `sds-local-volume-csi-node` Pod continues to run, this is most likely due to the presence on the node of LVMVolumeGroup resources that are used by one of the LocalStorageClass resources. The verification process is described above.

Absence of component service Pods on the desired node

The issue may be related to incorrectly set labels. The nodes used by the module are determined by the labels specified in the module settings in the `nodeSelector` field. To view the current labels, run:

```
d8 k get mc sds-local-volume -o=jsonpath={.spec.settings.dataNodes.nodeSelector}
```

Example output:

```
nodeSelector:
  my-custom-label-key: my-custom-label-value
```

Additionally, you can check the selectors used by the module in the secret configuration `d8-sds-local-volume-controller-config` in the `d8-sds-local-volume` namespace:

```
d8 k -n d8-sds-local-volume get secret d8-sds-local-volume-controller-config  -o
jsonpath='{.data.config}' | base64 --decode
```

Example output:

```
nodeSelector:
  kubernetes.io/os: linux
  my-custom-label-key: my-custom-label-value
```

This command output should list all the labels from the module settings `data.nodeSelector`, as well as `kubernetes.io/os: linux`.

Check the labels on the desired node:

```
d8 k get node %node-name% --show-labels
```

If necessary, add the missing labels to the desired node:

```
d8 k label node %node-name% my-custom-label-key=my-custom-label-value
```

If the labels are present, check for the presence of the label `storage.deckhouse.io/sds-local-volume-node=` on the node. If this label is missing, ensure that the `sds-local-volume-controller` is

running, and review its logs:

```
d8 k -n d8-sds-local-volume get po -l app=sds-local-volume-controller
d8 k -n d8-sds-local-volume logs -l app=sds-local-volume-controller
```

Data migration between PVCs

Copy the following script into a file named `migrate.sh` on any master node:

```
#!/bin/bash

ns=$1
src=$2
dst=$3

if [[ -z $3 ]]; then
  echo "You must give as args: namespace source_pvc_name destination_pvc_name"
  exit 1
fi

echo "Creating job yaml"
cat > migrate-job.yaml << EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: migrate-pv-$src
  namespace: $ns
spec:
  template:
    spec:
      containers:
      - name: migrate
        image: debian
        command: [ "/bin/bash", "-c" ]
        args:
          -
            apt-get update && apt-get install -y rsync &&
            ls -lah /src_vol /dst_vol &&
            df -h &&
            rsync -avPS --delete /src_vol/ /dst_vol/ &&
            ls -lah /dst_vol/ &&
            du -shxc /src_vol/ /dst_vol/
        volumeMounts:
        - mountPath: /src_vol
          name: src
          readOnly: true
        - mountPath: /dst_vol
          name: dst
      restartPolicy: Never
      volumes:
      - name: src
        persistentVolumeClaim:
          claimName: $src
      - name: dst
        persistentVolumeClaim:
          claimName: $dst
  backoffLimit: 1
EOF

d8 k create -f migrate-job.yaml
d8 k -n $ns get jobs -o wide
kubectl_completed_check=0
```

```
echo "Waiting for data migration to be completed"
while [[ $kubectl_completed_check -eq 0 ]]; do
   d8 k -n $ns get pods | grep migrate-pv-$src
   sleep 5
   kubectl_completed_check=`d8 k -n $ns get pods | grep migrate-pv-$src | grep "Completed" |
wc -l`
done
echo "Data migration completed"
```

To use the script, run the following command:

```
migrate.sh NAMESPACE SOURCE_PVC_NAME DESTINATION_PVC_NAME
```

Creating volume snapshots

You can read more about snapshots and associated resources in the Kubernetes documentation.

1. Enable the `snapshot-controller` module:

   ```
   d8 k apply -f -<<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: snapshot-controller
   spec:
     enabled: true
     version: 1
   EOF
   ```

2. Now you can create volume snapshots. To do this, execute the following command with the necessary parameters:

   ```
   d8 k apply -f -<<EOF
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshot
   metadata:
     name: my-snapshot
     namespace: <name of the namespace where the PVC is located>
   spec:
     volumeSnapshotClassName: sds-local-volume-snapshot-class
     source:
       persistentVolumeClaimName: <name of the PVC to snapshot>
   EOF
   ```

   Note that `sds-local-volume-snapshot-class` is created automatically, and its `deletionPolicy` is set to `Delete`, which means that the VolumeSnapshotContent resource will be deleted when the associated VolumeSnapshot resource is deleted.

3. To check the status of the created snapshot, execute the command:

   ```
   d8 k get volumesnapshot
   ```

   This command will display a list of all snapshots and their current status.

Additional resources

- Module `sds-local-volume` parameters
- Module `sds-local-volume` custom resources
- Module `sds-node-configurator` custom resources
- Module `snapshot-controller` documentation

Setting up replicated storage based on DRBD

Data replication across multiple nodes ensures fault tolerance and data availability, even if a hardware or software failure occurs on one of the nodes. This guarantees data preservation on other nodes, maintaining continuous access. Such a model is essential for critical data and distributed infrastructures requiring high availability and minimizing data loss during failures.

To create replicated block StorageClass objects based on Distributed Replicated Block Device (DRBD), the `sds-replicated-volume` module is used. It leverages LINSTOR as the backend.

Enabling the module

Discovery of LVM components

Before creating StorageClass objects based on Logical Volume Manager (LVM), it is necessary to detect the block devices and volume groups available on the nodes and obtain current information about their state. To do this, enable the `sds-node-configurator` module:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-node-configurator
spec:
  enabled: true
  version: 1
EOF
```

Wait for the `sds-node-configurator` module to reach the `Ready` status. To check the status, run the following command:

```
d8 k get modules sds-node-configurator -w
```

In the output, you should see information about the `sds-node-configurator` module:

```
NAME                      STAGE   SOURCE    PHASE       ENABLED    READY
sds-node-configurator             Embedded  Available   True       True
```

Connecting DRBD

To enable the `sds-replicated-volume` module with default settings, run the following command:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-replicated-volume
spec:
  enabled: true
  version: 1
EOF
```

This will install the DRBD kernel module on all cluster nodes, register the CSI driver, and launch the `sds-replicated-volume` component Pods.

Wait until the `sds-replicated-volume` module reaches the `Ready` status. To check the module status, run the following command:

```
d8 k get modules sds-replicated-volume -w
```

In the output, you should see information about the `sds-replicated-volume` module:

```
NAME                      STAGE   SOURCE    PHASE       ENABLED    READY
sds-replicated-volume             Embedded  Available   True       True
```

To check that all Pods in the `d8-sds-replicated-volume` and `d8-sds-node-configurator` namespaces are in the `Running` or `Completed` state and have been started on all nodes where DRBD resources are planned to be used, use the following commands:

```
d8 k -n d8-sds-replicated-volume get pod -w
d8 k -n d8-sds-node-configurator get pod -w
```

> **i** Avoid configuring the `LINSTOR` backend manually, as this can can result in errors.

Node pre-configuration

Creating LVM volume groups

Before configuring the creation of StorageClass objects, you need to combine the available block devices on the nodes into LVM volume groups. These volume groups will later be used to place PersistentVolumes. To get the available block devices, you can use the BlockDevice resource, which reflects their current state:

```
d8 k get bd
```

In the output, you should see a list of available block devices:

```
NAME                                             NODE       CONSUMABLE   SIZE
PATH
dev-ef4fb06b63d2c05fb6ee83008b55e486aa1161aa   worker-0   false        976762584Ki
/dev/nvme1n1
dev-0cfc0d07f353598e329d34f3821bed992c1ffbcd   worker-0   false        894006140416
/dev/nvme0n1p6
dev-7e4df1ddf2a1b05a79f9481cdf56d29891a9f9d0   worker-1   false        976762584Ki
/dev/nvme1n1
dev-b103062f879a2349a9c5f054e0366594568de68d   worker-1   false        894006140416
/dev/nvme0n1p6
dev-53d904f18b912187ac82de29af06a34d9ae23199   worker-2   false        976762584Ki
/dev/nvme1n1
dev-6c5abbd549100834c6b1668c8f89fb97872ee2b1   worker-2   false        894006140416
/dev/nvme0n1p6
```

In the example output, there are six block devices available across three nodes.

To combine the block devices on a node, you need to create an LVM volume group using the LVMVolumeGroup resource. To create the LVMVolumeGroup resource on node `worker-0`, apply the following resource, replacing the node and block device names with your own:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: LVMVolumeGroup
metadata:
  name: "vg-on-worker-0"
spec:
  type: Local
  local:
    # Replace with the name of the node where you are creating the volume group.
    nodeName: "worker-0"
  blockDeviceSelector:
    matchExpressions:
      - key: kubernetes.io/metadata.name
        operator: In
        values:
          # Replace with the names of the block devices for the node you are
creating the volume group on.
          - dev-ef4fb06b63d2c05fb6ee83008b55e486aa1161aa
          - dev-0cfc0d07f353598e329d34f3821bed992c1ffbcd
  # The name of the LVM volume group that will be created from the specified block
devices on the selected node.
  actualVGNameOnTheNode: "vg"
  # Uncomment if you want to be able to create thin pools.
  # thinPools:
  #   - name: thin-pool-0
  #     size: 70%
EOF
```

Wait for the created LVMVolumeGroup resource to reach the `Ready` phase. To check the resource phase, run the following command:

```
d8 k get lvg vg-on-worker-0 -w
```

In the output, you should see information about the resource phase:

```
NAME             THINPOOLS   CONFIGURATION APPLIED   PHASE   NODE       SIZE
ALLOCATED SIZE   VG   AGE
vg-on-worker-0   1/1         True                    Ready   worker-0   360484Mi
30064Mi          vg   1h
```

If the resource has transitioned to the `Ready` phase, this means that an LVM volume group named vg has been created on node `worker-0` from the block devices `/dev/nvme1n1` and `/dev/nvme0n1p6`.

Next, you need to repeat the creation of LVMVolumeGroup resources for the remaining nodes (`worker-1` and `worker-2`), changing the resource name, node name, and block device names accordingly.

Ensure that LVM volume groups have been created on all nodes where they are planned to be used by running the following command:

```
d8 k get lvg -w
```

In the output, you should see a list of created volume groups:

```
NAME             THINPOOLS   CONFIGURATION APPLIED   PHASE   NODE       SIZE
ALLOCATED SIZE   VG   AGE
vg-on-worker-0   0/0         True                    Ready   worker-0   360484Mi
30064Mi          vg   1h
vg-on-worker-1   0/0         True                    Ready   worker-1   360484Mi
30064Mi          vg   1h
vg-on-worker-2   0/0         True                    Ready   worker-2   360484Mi
30064Mi          vg   1h
```

Creating replicated thick pools

> **i** To work with thin pool, you need to enable the parameter enableThinProvisioning

Now that the necessary LVM volume groups are created on the nodes, you need to combine them into a single logical space. This can be done by combining them into replicated storage pools in the `LINSTOR` backend through the ReplicatedStoragePool resource interface.

Storage pools can be of two types: LVM (thick) and LVMThin (thin). The thick pool offers high performance, comparable to the performance of the storage device, but it does not allow snapshots. Example of creating a replicated thick pool:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: thick-pool
spec:
  type: LVM
  lvmVolumeGroups:
    - name: vg-1-on-worker-0
    - name: vg-1-on-worker-1
    - name: vg-1-on-worker-2
EOF
```

Wait for the created ReplicatedStoragePool resource to reach the `Completed` phase. To check the resource phase, run the following command:

```
d8 k get rsp data -w
```

In the output, you should see information about the resource phase:

```
NAME         PHASE       TYPE   AGE
thick-pool   Completed   LVM    87d
```

Creating replicated thin pools

Unlike thick pools, a thin pool allows the use of snapshots but has lower performance.

The previously created LVMVolumeGroup is suitable for creating thick pools. If you need the ability to create replicated thin pools, update the configuration of the LVMVolumeGroup resources by adding a definition for the thin pool:

```
d8 k patch lvg vg-on-worker-0 --type='json' -p='[
  {
    "op": "add",
    "path": "/spec/thinPools",
    "value": [
      {
        "name": "thin-pool-0",
        "size": "70%"
      }
    ]
  }
]'
```

In the updated version of LVMVolumeGroup, 70% of the available space will be used to create thin pools. The remaining 30% can be used for thick pools.

Repeat the addition of thin pools for the remaining nodes (`worker-1` and `worker-2`). Example of creating a replicated thin pool:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStoragePool
metadata:
  name: thin-pool
spec:
  type: LVMThin
  lvmVolumeGroups:
    - name: vg-1-on-worker-0
      thinPoolName: thin-pool-0
    - name: vg-1-on-worker-1
      thinPoolName: thin-pool-0
    - name: vg-1-on-worker-2
      thinPoolName: thin-pool-0
EOF
```

Wait for the created ReplicatedStoragePool resource to transition to the `Completed` phase. To check the resource phase, run the following command:

```
d8 k get rsp data -w
```

In the output, you should see information about the resource phase:

```
NAME        PHASE       TYPE      AGE
thin-pool   Completed   LVMThin   87d
```

Creating StorageClass objects

StorageClass objects are created through the ReplicatedStorageClass resource, which defines the configuration for the desired StorageClass. Manually creating a StorageClass resource without ReplicatedStorageClass may lead to undesired behavior.

Example of creating a ReplicatedStorageClass resource based on a thick pool, where the PersistentVolumes will be placed on volume groups across three nodes:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: ReplicatedStorageClass
metadata:
  name: replicated-storage-class
spec:
  # Specify the name of one of the storage pools created earlier.
  storagePool: thick-pool
  # Reclaim policy when deleting PVC.
  # Allowed values: "Delete", "Retain".
  # [Learn more...](https://kubernetes.io/docs/concepts/storage/persistent-
volumes/#reclaiming)
  reclaimPolicy: Delete
  # Replicas can be placed on any available node: no more than one replica of a
specific volume on one node.
  # The cluster does not have zones (no nodes with the topology.kubernetes.io/zone
labels).
  topology: Ignored
  # Replication mode where the volume remains available for read and write even if
one replica becomes unavailable.
  # Data is stored in three instances across different nodes.
  replication: ConsistencyAndAvailability
EOF
```

Check that the created ReplicatedStorageClass resource has transitioned to the `Created` phase by running the following command:

```
d8 k get rsc replicated-storage-class -w
```

In the output, you should see information about the created ReplicatedStorageClass resource:

```
NAME                        PHASE     AGE
replicated-storage-class    Created   1h
```

Check that the corresponding StorageClass has been generated by running the following command:

```
d8 k get sc replicated-storage-class
```

In the output, you should see information about the generated StorageClass:

```
NAME                        PROVISIONER                       RECLAIMPOLICY
VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
replicated-storage-class    local.csi.storage.deckhouse.io    Delete
WaitForFirstConsumer   true                   1h
```

If a StorageClass with the name `replicated-storage-class` appears, it means the configuration of the `sds-replicated-volume` module is complete. Users can now create PersistentVolume objects by specifying the `replicated-storage-class` StorageClass. With the above configuration, a volume with three replicas across different nodes will be created.

Additional resources

- Module `sds-node-configurator` documentation
- Module `sds-replicated-volume` documentation

Managing DRBD-based replicated storage

⚠ Functionality is guaranteed only if the system requirements are met. Using it under other conditions is possible, but stable operation is not guaranteed.

System requirements

i Applicable to both single-zone clusters and clusters that use multiple availability zones.

- Use stock kernels shipped with the supported distributions.
- For network connectivity, use infrastructure with bandwidth of 10 Gbps or higher.
- To achieve maximum performance, network latency between nodes should be within 0.5–1 ms.
- Do not use another SDS (Software-defined storage) to provide disks for Deckhouse SDS.

Recommendations

- Do not use RAID. See below for details.
- Use local physical disks. See below.
- For stable cluster operation (with reduced performance), allowable network latency between nodes must not exceed 20 ms.

Retrieving space-usage information

There are two ways to obtain this information:

1. Via the Grafana dashboard:

   Go to “Dashboards” → “Storage” → “LINSTOR/DRBD”. The current cluster space usage level is shown in the upper-right corner.

> **Warning.** The value reflects the state of all available space. When you create volumes with two replicas, divide the reported figures by two to estimate how many volumes can actually be placed.

2. Via the command line:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor storage-pool list
```

> **Warning.** The value reflects the state of all available space. When you create volumes with two replicas, divide the reported figures by two to estimate how many volumes can actually be placed.

Assigning the default StorageClass

To designate a StorageClass as the default, set the field `spec.isDefault` to `true` in the ReplicatedStorageClass custom resource.

Adding an existing LVMVolumeGroup

1. Assign the tag `storage.deckhouse.io/enabled=true` to the Volume Group:

   ```
   vgchange myvg-0 --add-tag storage.deckhouse.io/enabled=true
   ```

   After that the Volume Group will be discovered automatically and a corresponding LVMVolumeGroup resource will be created.

2. Reference this resource in ReplicatedStoragePool parameters using `spec.lvmVolumeGroups[].name`.
   If an LVMThin pool is used, additionally specify its name in `spec.lvmVolumeGroups[].thinPoolName`.

Changing DRBD volume limits and cluster ports

The default port range for DRBD resources is TCP `7000`–`7999`. You can override it with the `drbdPortRange` setting by specifying the desired `minPort` and `maxPort` values.

> ⚠ After changing `drbdPortRange`, restart the LINSTOR controller so that the new settings take effect. Existing DRBD resources will retain their assigned ports.

Correctly rebooting a node with DRBD resources

> To ensure stable module operation, avoid rebooting multiple nodes at the same time.

1. Drain the target node:

   ```
   d8 k drain test-node-1 --ignore-daemonsets --delete-emptydir-data
   ```

2. Make sure there are no faulty DRBD resources or resources in the `SyncTarget` state. To check this, run the following command and analyze the output:

   ```
   d8 k -n d8-sds-replicated-volume exec -t deploy/linstor-controller -- linstor r l --faulty
   ```

   If any resources are in the `SyncTarget` state, wait for the synchronization to complete or take corrective actions.

   Example output:

```
Defaulted container "linstor-controller" out of: linstor-controller, kube-rbac-
proxy
+-------------------------------------------------------------+
| ResourceName | Node | Port | Usage | Conns | State | CreatedOn |
|=============================================================|
+-------------------------------------------------------------+
```

3. Reboot the node and wait for all DRBD resources to resynchronize, then run `uncordon`:

```
d8 k uncordon test-node-1
node/test-node-1 uncordoned
```

If you need to reboot another node, repeat the procedure.

Moving resources to free up space in a Storage Pool

1. List the Storage Pools on the source node to identify which one is running low on free space:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor storage-pool list -n
OLD_NODE
```

2. Determine which volumes are located in the overloaded Storage Pool to identify potential candidates for migration:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor volume list -n
OLD_NODE
```

3. Get the list of resources that own these volumes so that you can proceed with moving them:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor resource list-volumes
```

4. Create copies of the selected resources on another node (no more than 1–2 at a time):

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor --yes-i-am-sane-
and-i-understand-what-i-am-doing resource create NEW_NODE RESOURCE_NAME
```

5. Wait for the resource synchronization to complete to ensure the data has been replicated properly:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor resource-definition
wait-sync RESOURCE_NAME
```

6. Remove the resource from the source node to free up space in the overloaded Storage Pool:

```
d8 k exec -n d8-sds-replicated-volume deploy/linstor-controller -- linstor --yes-i-am-sane-
and-i-understand-what-i-am-doing resource delete OLD_NODE RESOURCE_NAME
```

Automatic management of replicas and monitoring of LINSTOR state

Replica management and state monitoring are automated in the `replicas_manager.sh` script. It checks the availability of the LINSTOR controller, identifies faulty or corrupted resources, creates database backups, and manages disk replicas, including configuring `TieBreaker` for quorum.

To check the existence of the `replicas_manager.sh` script, run the following command on any master node:

```
ls -l /opt/deckhouse/sbin/replicas_manager.sh
```

Upon execution, the script performs the following actions:

- Verifies the availability of the controller and connectivity to satellites.

- Identifies faulty or corrupted resources.
- Creates a backup of the database.
- Manages the number of disk replicas, adding new ones as needed.
- Configures TieBreaker for resources with two replicas.
- Logs all actions to a file named `linstor_replicas_manager_<date_time>.log`.
- Provides recommendations for resolving issues, such as stuck replicas.

Configuration variables for `replicas_manager.sh`:

- `NON_INTERACTIVE`: Enables non-interactive mode.
- `TIMEOUT_SEC`: Timeout between attempts, in seconds (default: 10).
- `EXCLUDED_RESOURCES_FROM_CHECK`: Regular expression to exclude resources from checks.
- `CHUNK_SIZE`: Chunk size for processing resources (default: 10).
- `NODE_FOR_EVICT`: The name of the node excluded from creating replicas.
- `LINSTOR_NAMESPACE`: Kubernetes namespace (default: `d8-sds-replicated-volume`).
- `DISKLESS_STORAGE_POOL`: Pool for diskless replicas (default: `DfltDisklessStorPool`).

Evicting DRBD resources from a node

Eviction of DRBD resources from a node is performed using the `evict.sh` script. It can operate in two modes:

- Node removal: Additional replicas are created for every resource, after which the node is removed from LINSTOR and Kubernetes.
- Resource removal: Replicas are created for the resources, after which the resources themselves are removed from LINSTOR (the node remains in the cluster).

Preparing and running the script

Before eviction:

1. Make sure the script is present on the master node:

   ```
   ls -l /opt/deckhouse/sbin/evict.sh
   ```

2. Fix faulty resources:

   ```
   d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -- linstor resource list --faulty
   ```

3. Ensure all Pods in the `d8-sds-replicated-volume` namespace are in the `Running` state:

   ```
   d8 k -n d8-sds-replicated-volume get pods | grep -v Running
   ```

Example of removing a node from LINSTOR and Kubernetes

Run `evict.sh` on any master node in interactive mode, specifying `--delete-node`:

```
/opt/deckhouse/sbin/evict.sh --delete-node
```

For non-interactive mode, add `--non-interactive` and specify the node name. The script will perform all actions without prompting:

```
/opt/deckhouse/sbin/evict.sh --non-interactive --delete-node --node-name "worker-1"
```

Example of removing resources from a node

Run `evict.sh` on any master node in interactive mode, specifying `--delete-resources-only`:

```
/opt/deckhouse/sbin/evict.sh --delete-resources-only
```

For non-interactive mode, add `--non-interactive` and specify the node name. In this mode, the script runs through all actions without asking for confirmation:

```
/opt/deckhouse/sbin/evict.sh --non-interactive --delete-resources-only --node-name
"worker-1"
```

⚠ After the script completes, the node remains in `SchedulingDisabled` status and LINSTOR sets the property `AutoplaceTarget=false`. This blocks automatic creation of new resources on the node.

To allow resource placement again, run:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -
- linstor'
linstor node set-property "worker-1" AutoplaceTarget
d8 k uncordon "worker-1"
```

Check the `AutoplaceTarget` parameter on all nodes (the field will be empty on nodes where LINSTOR resource placement is allowed):

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -
- linstor'
linstor node list -s AutoplaceTarget
```

Parameters of the evict.sh script

- `--delete-node`: Remove a node from LINSTOR and Kubernetes after creating additional replicas for all resources on the node.
- `--delete-resources-only`: Remove resources from the node without deleting the node from LINSTOR and Kubernetes, after creating additional replicas.
- `--non-interactive`: Run the script in non-interactive mode.
- `--node-name`: The name of the node to evict resources from. Mandatory when using `--non-interactive`.
- `--skip-db-backup`: Skip creating a LINSTOR DB backup before operations.
- `--ignore-advise`: Perform operations despite `linstor advise resource` warnings. Use if the script was interrupted and the number of replicas for some resources does not match the value specified in the `ReplicatedStorageClass`.
- `--exclude-resources-from-check`: Exclude resources listed with `|` from checks.

Troubleshooting

Issues can arise at various component layers. The cheat sheet below helps diagnose volume failures in LINSTOR.

Start error of linstor-node while loading the DRBD module

1. Check the `linstor-node` Pods:

   ```
   d8 k get pod -n d8-sds-replicated-volume -l app=linstor-node
   ```

2. If some Pods are stuck in `Init`, check the DRBD version and bashible logs on the node:

   ```
   cat /proc/drbd
   journalctl -fu bashible
   ```

Most likely causes:

- DRBDv8 is loaded instead of the required DRBDv9. Verify the version:

  ```
  cat /proc/drbd
  ```

  If the `/proc/drbd` file is missing, the module is not loaded and the problem lies elsewhere.

- Secure Boot is enabled. Because DRBD is built dynamically (similar to dkms) without a digital

signature, the module is not supported when Secure Boot is enabled.

FailedMount error when starting a Pod

Pod stuck in ContainerCreating

If the Pod is stuck in `ContainerCreating` and `d8 k describe pod` shows errors like the one below, the device is mounted on another node:

```
rpc error: code = Internal desc = NodePublishVolume failed for pvc-b3e51b8a-9733-
4d9a-bf34-84e0fee3168d: checking
for exclusive open failed: wrong medium type, check device health
```

Check where the device is in use:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -
- linstor'
linstor resource list -r pvc-b3e51b8a-9733-4d9a-bf34-84e0fee3168d
```

The `InUse` flag shows on which node the device is used; unmount the disk manually on that node.

Input/output error

Such errors usually occur during filesystem creation (mkfs). Check `dmesg` on the node where the Pod is starting:

```
dmesg | grep 'Remote failed to finish a request within'
```

If the output contains messages like `Remote failed to finish a request within …`, the disk subsystem may be too slow for DRBD to operate correctly.

After the ReplicatedStoragePool resource is deleted, the corresponding Storage Pool remains in the backend

This is expected behavior. The `sds-replicated-volume` module does not handle deletion operations for the ReplicatedStoragePool resource.

Restrictions on changing ReplicatedStorageClass spec

Only the `isDefault` field can be modified. All other parameters are immutable — this is expected behavior.

Deleting a child StorageClass when deleting a ReplicatedStorageClass

If the StorageClass is in the `Created` status, it can be deleted. For other statuses you must restore the resource or delete the StorageClass manually.

Errors when creating a Storage Pool or StorageClass

For temporary external issues (for example, when `kube-apiserver` is unavailable) the module automatically retries the failed operation.

Error “You’re not allowed to change state of linstor cluster manually”

Operations requiring manual intervention are partially or fully automated in the `sds-replicated-volume` module. Therefore, the module restricts the list of allowed LINSTOR commands. For example, creating a Tie-Breaker is automated because LINSTOR sometimes does not create one for two-replica resources. To see the list of allowed commands, run:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -
- linstor'
linstor --help
```

Restoring the database from a backup

Backend resource backups are stored in Secrets as YAML files split into segments for easier restoration. Backups are created automatically on a schedule.

A correctly formed backup looks like this:

```
linstor-20240425074718-backup-0              Opaque                                 1
28s     sds-replicated-volume.deckhouse.io/linstor-db-backup=20240425074718
linstor-20240425074718-backup-1              Opaque                                 1
28s     sds-replicated-volume.deckhouse.io/linstor-db-backup=20240425074718
linstor-20240425074718-backup-2              Opaque                                 1
28s     sds-replicated-volume.deckhouse.io/linstor-db-backup=20240425074718
linstor-20240425074718-backup-completed      Opaque                                 0
28s     <none>
```

The backup is stored in encoded segments in Secrets named `linstor-%date_time%-backup-{0..2}`. The Secret `linstor-%date_time%-backup-completed` contains no data and serves as a marker that the backup process completed successfully.

Backup restoration procedure

1. Set environment variables:

   ```
   NAMESPACE="d8-sds-replicated-volume"
   BACKUP_NAME="linstor_db_backup"
   ```

2. List available backups:

   ```
   d8 k -n $NAMESPACE get secrets --show-labels
   ```

   Example output:

```
linstor-20240425072413-backup-0             Opaque                          1      33m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072413
linstor-20240425072413-backup-1             Opaque                          1      33m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072413
linstor-20240425072413-backup-2             Opaque                          1      33m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072413
linstor-20240425072413-backup-completed     Opaque                          0      33m     <none>
linstor-20240425072510-backup-0             Opaque                          1      32m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072510
linstor-20240425072510-backup-1             Opaque                          1      32m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072510
linstor-20240425072510-backup-2             Opaque                          1      32m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072510
linstor-20240425072510-backup-completed     Opaque                          0      32m     <none>
linstor-20240425072634-backup-0             Opaque                          1      31m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072634
linstor-20240425072634-backup-1             Opaque                          1      31m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072634
linstor-20240425072634-backup-2             Opaque                          1      31m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072634
linstor-20240425072634-backup-completed     Opaque                          0      31m     <none>
linstor-20240425072918-backup-0             Opaque                          1      28m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072918
linstor-20240425072918-backup-1             Opaque                          1      28m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072918
linstor-20240425072918-backup-2             Opaque                          1      28m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425072918
linstor-20240425072918-backup-completed     Opaque                          0      28m     <none>
linstor-20240425074718-backup-0             Opaque                          1      10m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425074718
linstor-20240425074718-backup-1             Opaque                          1      10m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425074718
linstor-20240425074718-backup-2             Opaque                          1      10m     sds-
replicated-volume.deckhouse.io/linstor-db-backup=20240425074718
linstor-20240425074718-backup-completed     Opaque                          0      10m     <none>
```

3. Each backup has its own creation-time label. Choose the desired one and save it to an environment variable. In this example we use the most recent label:

   ```
   LABEL_SELECTOR="sds-replicated-volume.deckhouse.io/linstor-db-
   backup=20240425074718"
   ```

4. Create a temporary directory to store archive parts:

   ```
   TMPDIR=$(mktemp -d)
   echo "Temporary directory: $TMPDIR"
   ```

5. Create an empty archive and combine Secret data into one file:

   ```
   COMBINED="${BACKUP_NAME}_combined.tar"
   > "$COMBINED"
   ```

6. Get the list of Secrets by label, decode the data, and append it to the archive:

   ```
   MOBJECTS=$(d8 k get rsmb -l "$LABEL_SELECTOR" --sort-by=.metadata.name -o jsonpath="
   {.items[*].metadata.name}")

   for MOBJECT in $MOBJECTS; do
     echo "Process: $MOBJECT"
    d8 k get rsmb "$MOBJECT" -o jsonpath="{.data}" | base64 --decode >> "$COMBINED"
   done
   ```

7. Extract the archive to get backup files:

```
mkdir -p "./backup"
tar -xf "$COMBINED" -C "./backup" --strip-components=2
```

8. Check the backup contents:

   ```
   ls ./backup
   TMPDIR=$(mktemp -d)
   echo "Temporary directory: $TMPDIR"
   ```

   Example output:

   ```
   ebsremotes.yaml                     layerdrbdvolumedefinitions.yaml
   layerwritecachevolumes.yaml  propscontainers.yaml      satellitescapacity.yaml
   secidrolemap.yaml         trackingdate.yaml
   files.yaml                          layerdrbdvolumes.yaml
   linstorremotes.yaml          resourceconnections.yaml  schedules.yaml
   secobjectprotection.yaml  volumeconnections.yaml
   keyvaluestore.yaml                  layerluksvolumes.yaml
   linstorversion.yaml          resourcedefinitions.yaml  secaccesstypes.yaml
   secroles.yaml             volumedefinitions.yaml
   layerbcachevolumes.yaml             layeropenflexresourcedefinitions.yaml
   nodeconnections.yaml         resourcegroups.yaml       secaclmap.yaml
   sectyperules.yaml         volumegroups.yaml
   layercachevolumes.yaml              layeropenflexvolumes.yaml
   nodenetinterfaces.yaml       resources.yaml            secconfiguration.yaml
   sectypes.yaml             volumes.yaml
   layerdrbdresourcedefinitions.yaml  layerresourceids.yaml
   nodes.yaml                   rollback.yaml             secdfltroles.yaml
   spacehistory.yaml
   layerdrbdresources.yaml             layerstoragevolumes.yaml
   nodestorpool.yaml            s3remotes.yaml            secidentities.yaml
   storpooldefinitions.yaml
   ```

9. Restore the required entity by applying the corresponding YAML file:

   ```
   d8 k apply -f %something%.yaml
   ```

   Or bulk-apply for a full restore:

   ```
   d8 k apply -f ./backup/
   ```

Missing sds-replicated-volume service Pods on a selected node

The issue is most likely related to node labels.

- Check `dataNodes.nodeSelector` in module settings:

  ```
  d8 k get mc sds-replicated-volume -o=jsonpath={.spec.settings.dataNodes.nodeSelector}
  ```

- Check the selectors used by `sds-replicated-volume-controller`:

  ```
  d8 k -n d8-sds-replicated-volume get secret d8-sds-replicated-volume-controller-config  -o
  jsonpath='{.data.config}' | base64 --decode
  ```

- The secret `d8-sds-replicated-volume-controller-config` must contain the selectors specified in module settings plus `kubernetes.io/os: linux`.
- Verify that the all labels from the `d8-sds-replicated-volume-controller-config` secret are present on the node:

```
d8 k get node worker-0 --show-labels
```

- If the labels are missing, add them via NodeGroup templates or directly to the node.
- If the labels exist, check whether the node has the label `storage.deckhouse.io/sds-replicated-volume-node=`.
  If not, check whether `sds-replicated-volume-controller` is running and inspect its logs:

```
d8 k -n d8-sds-replicated-volume get po -l app=sds-replicated-volume-controller
d8 k -n d8-sds-replicated-volume logs -l app=sds-replicated-volume-controller
```

Additional support

Reasons for failed operations are shown in the `status.reason` field of ReplicatedStoragePool and ReplicatedStorageClass resources. If that information is insufficient, consult the `sds-replicated-volume-controller` logs.

Migration from the linstor module to sds-replicated-volume

During migration the LINSTOR control plane and its CSI are temporarily unavailable, which can affect PV operations (creation, expansion, or deletion).

⚠ User data is not affected because the migration moves to a new namespace and adds components that manage volumes.

Migration procedure

1. Ensure no faulty resources exist in the backend. The command should return an empty list:

   ```
   alias linstor='d8 k -n d8-linstor exec -ti deploy/linstor-controller -- linstor'
   linstor resource list --faulty
   ```

   i **Warning.** All resources must be healthy before migration.

2. Disable the `linstor` module:

   ```
   d8 k patch moduleconfig linstor --type=merge -p '{"spec": {"enabled": false}}'
   ```

3. Wait until the `d8-linstor` namespace is deleted:

   ```
   d8 k get namespace d8-linstor
   ```

4. Create a ModuleConfig for `sds-node-configurator`:

   ```
   d8 k apply -f -<<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: sds-node-configurator
   spec:
     enabled: true
     version: 1
   EOF
   ```

5. Wait until `sds-node-configurator` reaches `Ready`:

   ```
   d8 k get moduleconfig sds-node-configurator
   ```

6. Create a ModuleConfig for `sds-replicated-volume`:

   > **Warning.** If `settings.dataNodes.nodeSelector` is not specified for `sds-replicated-volume`, its value will be taken from the `linstor` module. If it is absent there as well, it will remain empty and all cluster nodes will be considered data nodes.

   ```
   d8 k apply -f - <<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: sds-replicated-volume
   spec:
     enabled: true
     version: 1
   EOF
   ```

7. Wait until `sds-replicated-volume` reaches `Ready`:

   ```
   d8 k get moduleconfig sds-replicated-volume
   ```

8. Check the `sds-replicated-volume` settings:

   ```
   d8 k get moduleconfig sds-replicated-volume -oyaml
   ```

9. Wait until all Pods in the `d8-sds-replicated-volume` and `d8-sds-node-configurator` namespaces are `Ready` or `Completed`:

   ```
   d8 k get po -n d8-sds-node-configurator
   d8 k get po -n d8-sds-replicated-volume
   ```

10. Update the `linstor` alias and check resources:

    ```
    alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-controller -- linstor'
    linstor resource list --faulty
    ```

If no faulty resources are found, migration was successful.

Migrating to ReplicatedStorageClass

StorageClasses in this module are managed via the ReplicatedStorageClass resource. StorageClasses must not be created manually.

When migrating from the LINSTOR module, delete old StorageClasses and create new ones via ReplicatedStorageClass according to the table below.

Note that in old StorageClasses you look at the option in the `parameters` section of the StorageClass itself, while when creating a new one you set the corresponding option in ReplicatedStorageClass.

StorageClass parameter	ReplicatedStorageClass	Default	Notes
linstor.csi.linbit.com/placementCount: “1”	replication: “None”		One data replica will be created
linstor.csi.linbit.com/placementCount: “2”	replication: “Availability”		Two data replicas will be created
linstor.csi.linbit.com/placementCount: “3”	replication: “ConsistencyAndAvailability”	Yes	Three data replicas will be created
linstor.csi.linbit.com/storagePool: “name”	storagePool: “name”		Name of the storage pool used for storage

StorageClass parameter	ReplicatedStorageClass	Default	Notes
linstor.csi.linbit.com/allowRemoteVolumeAccess: “false”	volumeAccess: “Local”		Remote Pod access to data volumes is forbidden (local disk access within the node only)

Additional parameters:

- `reclaimPolicy` (Delete, Retain): Corresponds to `reclaimPolicy` of the old StorageClass.
- `zones`: List of zones to place resources in (direct cloud zone names). Note that remote Pod access to the data volume is possible only within one zone.
- `volumeAccess` values: `Local` (access strictly within the node), `EventuallyLocal` (a data replica will synchronize to the node after the Pod starts), `PreferablyLocal` (remote Pod access allowed, `volumeBindingMode: WaitForFirstConsumer`), `Any` (remote Pod access allowed, `volumeBindingMode: Immediate`).
- If you need `volumeBindingMode: Immediate`, set `volumeAccess` in ReplicatedStorageClass to `Any`.

Migrating to ReplicatedStoragePool

The ReplicatedStoragePool resource allows you to create Storage Pools in the backend. It is recommended to create this resource even for existing Storage Pools and reference existing LVMVolumeGroup. The controller will detect that the Storage Pools already exist and leave them unchanged, showing `Created` in `status.phase`.

Migration from the sds-drbd module to sds-replicated-volume

During migration the module control plane and its CSI are unavailable. This prevents creation, expansion, or deletion of PVs and the creation or deletion of Pods that use DRBD PVs for the duration of the migration.

⚠ The migration will not affect user data, as it is performed in a new namespace and volume management will be handled by new components that will replace the functionality of the previous module.

Migration procedure

1. Make sure there are no faulty DRBD resources in the cluster. The command should return an empty list:

   ```
   alias linstor='d8 k -n d8-sds-drbd exec -ti deploy/linstor-controller -- linstor'
   linstor resource list --faulty
   ```

   Warning. All DRBD resources must be healthy before migration.

2. Disable the `sds-drbd` module:

   ```
   d8 k patch moduleconfig sds-drbd --type=merge -p '{"spec": {"enabled": false}}'
   ```

3. Wait until the `d8-sds-drbd` namespace is deleted:

   ```
   d8 k get namespace d8-sds-drbd
   ```

4. Create a ModuleConfig for `sds-replicated-volume`:

   Warning. If `settings.dataNodes.nodeSelector` is not specified for `sds-replicated-volume`, its value will be taken from the `sds-drbd` module. If it is absent there as well, it will remain empty and all cluster nodes will be considered data nodes.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: sds-replicated-volume
spec:
  enabled: true
  version: 1
EOF
```

5. Wait until `sds-replicated-volume` reaches `Ready`:

```
d8 k get moduleconfig sds-replicated-volume
```

6. Check the `sds-replicated-volume` settings:

```
d8 k get moduleconfig sds-replicated-volume -oyaml
```

7. Wait until all Pods in the `d8-sds-replicated-volume` namespace are `Ready` or `Completed`:

```
d8 k get po -n d8-sds-replicated-volume
```

8. Update the `linstor` alias and check DRBD resources:

```
alias linstor='d8 k -n d8-sds-replicated-volume exec -ti deploy/linstor-
controller -- linstor'
linstor resource list --faulty
```

If no faulty resources are found, migration was successful.

> **Warning.** DRBDStoragePool and DRBDStorageClass resources will be automatically migrated to ReplicatedStoragePool and ReplicatedStorageClass. No user action is required.

The logic of these resources remains unchanged. However, verify that no DRBDStoragePool or DRBDStorageClass resources remain. If they do, contact the Deckhouse technical support.

Reasons to avoid RAID with sds-replicated-volume

Using DRBD with more than one replica already provides network-level RAID functionality. Local RAID can cause the following issues:

- Significantly increases space overhead when using redundant RAID.
  Example: ReplicatedStorageClass with `replication` set to `ConsistencyAndAvailability`.
  DRBD will store three data replicas (one per node). If those nodes use RAID1, storing 1 GB of data will require 6 GB of disk space. Redundant RAID is reasonable only to simplify server maintenance when storage cost is irrelevant. RAID1 then allows replacing disks without moving data replicas off a “problem” disk.
- With RAID0 the performance gain is negligible because data replication occurs over the network and the bottleneck is likely the network. Additionally, reduced host storage reliability can lead to data unavailability since DRBD failover from a broken replica to a healthy one is not instantaneous.

Recommendations for using local disks

DRBD uses the network for data replication. When using NAS, network load increases dramatically because nodes synchronize data not only with the NAS but also with each other. Latency for reads or writes also increases. NAS typically uses RAID on its side, adding further overhead.

Manual trigger the certificate renewal process

Although the certificate renewal process is automated, manual renewal might still be necessary because it can be performed during a convenient maintenance window when it is acceptable to restart

the module's objects. The automated renewal does not restart any objects.

To manually trigger the certificate renewal process, create a `ConfigMap` named `manualcertrenewal-trigger`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: manualcertrenewal-trigger
  namespace: d8-sds-replicated-volume
```

The system will stop all necessary module objects, update the certificates, and then restart them.

You can check the operation status using the following command:

```
d8 k -n d8-sds-replicated-volume get cm manualcertrenewal-trigger -ojsonpath='{.data.step}'
```

Possible statuses:

- `Prepared`: Health checks have passed successfully, and the downtime window has started.
- `TurnedOffAndRenewedCerts`: The system has been stopped and certificates have been renewed.
- `TurnedOn`: The system has been restarted.
- `Done`: The operation is complete and ready to be repeated.

Certificates are issued for a period of one year and are marked as expiring 30 days before their expiration date. The monitoring system alerts about expiring certificates using the `D8LinstorCertificateExpiringIn30d` alert.

To repeat the operation, simply remove the label from the trigger using the following command:

```
d8 k -n d8-sds-replicated-volume label cm manualcertrenewal-trigger storage.deckhouse.io/sds-replicated-volume-manualcertrenewal-completed-
```

Additional resources

- Module `sds-local-volume` parameters
- Module `sds-node-configurator` documentation
- Module `sds-replicated-volume` documentation

Distributed Ceph storage

Ceph is a scalable distributed storage system that ensures high availability and fault tolerance of data. Deckhouse Kubernetes Platform (DKP) provides Ceph cluster integration using the `csi-ceph` module. This enables dynamic storage management and the use of StorageClass based on RADOS Block Device (RBD) or CephFS.

> **i** The snapshot-controller module is required for working with snapshots.

This page provides instructions on connecting Ceph to Deckhouse, configuring authentication, creating StorageClass objects, and verifying storage functionality.

Migration from `ceph-csi` module

When switching from the `ceph-csi` module to `csi-ceph`, an automatic migration is performed, but its execution requires preliminary preparation:

1. Set the replica count to zero for all operators (redis, clickhouse, kafka, etc.). Exception: the `prometheus` operator will be disabled automatically.
2. Disable the `ceph-csi` module and enable `csi-ceph`.
3. Wait for the operation to complete. The Deckhouse logs should show the message "Finished migration from Ceph CSI module".
4. Verify functionality. Create test pods and PVCs to test CSI.
5. Restore the operators to their normal replica count.

⚠ If Ceph StorageClass was created without using the `CephCSIDriver` resource, manual migration will be required. Contact technical support.

Connecting to a Ceph cluster

To connect to a Ceph cluster, follow the step-by-step instructions below. Execute all commands on a machine with administrative access to the Kubernetes API.

1. Execute the command to activate the `csi-ceph` module:

   ```
   d8 s module enable csi-ceph
   ```

2. Wait for the module to transition to `Ready` state:

   ```
   d8 k get module csi-ceph -w
   ```

3. Ensure that all pods in the `d8-csi-ceph` namespace are in `Running` or `Completed` state and deployed on all cluster nodes:

   ```
   d8 k -n d8-csi-ceph get pod -owide -w
   ```

4. To configure the connection to the Ceph cluster, apply the CephClusterConnection resource.

   Example command:

   ```
   d8 k apply -f - <<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: CephClusterConnection
   metadata:
     name: ceph-cluster-1
   spec:
     # FSID/UUID of the Ceph cluster.
     # Get the FSID/UUID of the Ceph cluster using the command `ceph fsid`.
     clusterID: 2bf085fc-5119-404f-bb19-820ca6a1b07e
     # List of IP addresses of ceph-mon in the format 10.0.0.10:6789.
     monitors:
       - 10.0.0.10:6789
     # Username without `client.`.
     # Get the username using the command `ceph auth list`.
     userID: admin
     # Authorization key corresponding to userID.
     # Get the authorization key using the command `ceph auth get-key
   client.admin`.
     userKey: AQDiVXVmBJVRLxAAg65PhODrtwbwSWrjJwssUg==
   EOF
   ```

5. Verify the connection creation with the command (`Phase` should be in `Created` status):

   ```
   d8 k get cephclusterconnection ceph-cluster-1
   ```

6. Create a StorageClass object using the CephStorageClass resource. Manual creation of StorageClass without using CephStorageClass may lead to errors.

   Example of creating StorageClass based on RBD:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: CephStorageClass
metadata:
  name: ceph-rbd-sc
spec:
  clusterConnectionName: ceph-cluster-1
  reclaimPolicy: Delete
  type: RBD
  rbd:
    defaultFSType: ext4
    pool: ceph-rbd-pool
EOF
```

Example of creating StorageClass based on Ceph filesystem:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: CephStorageClass
metadata:
  name: ceph-fs-sc
spec:
  clusterConnectionName: ceph-cluster-1
  reclaimPolicy: Delete
  type: CephFS
  cephFS:
    fsName: cephfs
EOF
```

7. Verify that the created CephStorageClass resources have transitioned to `Created` state:

   ```
   d8 k get cephstorageclass
   ```

   This will output information about the created CephStorageClass resources:

   ```
   NAME          PHASE     AGE
   ceph-rbd-sc   Created   1h
   ceph-fs-sc    Created   1h
   ```

8. Verify the created StorageClass:

   ```
   d8 k get sc
   ```

   This will output information about the created StorageClass:

   ```
   NAME          PROVISIONER        RECLAIMPOLICY   VOLUMEBINDINGMODE
   ALLOWVOLUMEEXPANSION   AGE
   ceph-rbd-sc   rbd.csi.ceph.com   Delete          WaitForFirstConsumer   true
   15s
   ceph-fs-sc    rbd.csi.ceph.com   Delete          WaitForFirstConsumer   true
   15s
   ```

Ceph cluster connection setup is complete. You can use the created StorageClass to create PersistentVolumeClaim in your applications.

Additional information

How to get a list of RBD volumes separated by nodes

For monitoring and diagnostics, it’s useful to know which RBD volumes are connected to each cluster node. The following command provides detailed information about volume mapping:

```
d8 k -n d8-csi-ceph get po -l app=csi-node-rbd -o custom-
columns=NAME:.metadata.name,NODE:.spec.nodeName --no-headers \
  | awk '{print "echo "$2"; kubectl -n d8-csi-ceph exec  "$1" -c node -- rbd
showmapped"}' | bash
```

Which versions of Ceph clusters are supported

The `csi-ceph` module has specific requirements for the Ceph cluster version to ensure compatibility and stable operation. Officially supported versions are >= 16.2.0. In practice, the current version works with clusters running versions >= 14.2.0, but it’s recommended to update Ceph to the latest version.

Which volume access modes are supported

Different types of Ceph storage support different volume access modes, which is important to consider when planning application architecture.

- **RBD**: Supports only ReadWriteOnce (RWO) — access to volume from only one cluster node.
- **CephFS**: Supports ReadWriteOnce (RWO) and ReadWriteMany (RWX) — simultaneous access to volume from multiple cluster nodes.

Examples of permissions (caps) for Ceph users

To ensure proper operation of the `csi-ceph` module, Ceph users must have appropriate permissions (caps) configured. The required permissions depend on the storage type being used (RBD, CephFS, or both). Below are examples of correct permission configurations for different scenarios.

RBD

For a single pool named `rbd`, the following permissions are required:

```
[client.name]
       key = key
       caps mgr = "profile rbd pool=rbd"
       caps mon = "profile rbd"
       caps osd = "profile rbd pool=rbd"
```

CephFS

Before configuring CephFS permissions, ensure that a subvolumegroup `csi` (or another one specified in `Custom resources`) is created in CephFS.

You can create a new subvolumegroup using the following command on the Ceph management node:

```
ceph fs subvolumegroup create <fs_name> <group_name>
```

For example, to create a subvolumegroup `csi` for filesystem `myfs`:

```
ceph fs subvolumegroup create myfs csi
```

Required permissions for CephFS named `myfs`:

```
[client.name]
       key = key
       caps mds = "allow rwps fsname=myfs"
       caps mgr = "allow rw"
       caps mon = "allow r fsname=myfs"
       caps osd = "allow rw tag cephfs data=myfs, allow rw tag cephfs metadata=myfs"
```

CephFS + RBD

For a user that needs access to both CephFS `myfs` and RBD pool `rbd`, combine the permissions as follows:

```
[client.name]
       key = key
       caps mds = "allow rwps fsname=myfs"
       caps mgr = "allow rw,profile rbd pool=rbd"
       caps mon = "allow r fsname=myfs,profile rbd"
       caps osd = "allow rw tag cephfs metadata=myfs, allow rw tag cephfs
data=myfs,profile rbd pool=rbd"
```

Additional resources

- Module `csi-ceph` custom resources
- Module `snapshot-controller` documentation

HPE data storage

Deckhouse includes support for HPE 3PAR storage systems, enabling volume management in Kubernetes using a CSI driver. This integration provides reliable, scalable, and high-performance storage suitable for mission-critical workloads. The `csi-hpe` module is used to work with HPE 3PAR systems, allowing StorageClass creation in Kubernetes through the HPEStorageClass resource.

> ⚠ User-created StorageClass for the `csi.hpe.com` CSI driver is not allowed.
> Only HPE 3PAR storage systems are supported. For other HPE storage systems, contact the Deckhouse technical support.

This page provides instructions on connecting HPE 3PAR to Deckhouse, configuring the connection, creating StorageClass, and verifying storage functionality.

System requirements

- A deployed and configured HPE storage system.
- Unique IQNs in `/etc/iscsi/initiatorname.iscsi` on each Kubernetes node.

Configuration

Note that all commands must be run on a machine that has administrator access to the Kubernetes API.

Enabling the module

Enable the `csi-hpe` module. This will result in the following actions across all cluster nodes:

- Registration of the CSI driver.
- Launch of service Pods for the `csi-hpe` components.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-hpe
spec:
  enabled: true
  version: 1
EOF
```

Wait until the module transitions to the `Ready` state. Check the module's status with the following command:

```
d8 k get module csi-hpe -W
```

Creating a StorageClass

To create a StorageClass, you need to use the HPEStorageClass and HPEStorageConnection resource. Here is an example command to create such a resource:

- Creating a HPEStorageConnection resource:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: HPEStorageConnection
metadata:
  name: hpe
spec:
  controlPlane:
    backendAddress: "172.17.1.55" # Storage system address (mutable).
    username: "3paradm" # API username (mutable).
    password: "3pardata" # API password (mutable).
    serviceName: "primera3par-csp-svc"
    servicePort: "8080"
EOF
```

  Verify the creation of the object using the following command (`Phase` should be `Created`):

```
d8 k get hpestorageconnections.storage.deckhouse.io <hpestorageconnection-name>
```

- Creating a HPEStorageClass resource:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: HPEStorageClass
metadata:
  name: hpe
spec:
  pool: "test-cpg"
  accessProtocol: "iscsi" # fc or iscsi (iscsi by default), immutable.
  fsType: "xfs" # xfs, ext3, ext4, btrfs (ext4 by default), mutable.
  storageConnectionName: "hpe" # Immutable.
  reclaimPolicy: Delete # Delete or Retain.
  cpg: "test-cpg"
EOF
```

  Verify the creation of the object using the following command (`Phase` should be `Created`):

```
d8 k get hpestorageclasses.storage.deckhouse.io <hpestorageclass-name>
```

Module health verification

To verify module health, ensure that all Pods in the `d8-csi-hpe` namespace are in the `Running` or `Completed` state and are running on every node in the cluster:

```
d8 k -n d8-csi-hpe get pod -owide -w
```

Additional resources

- Module `csi-hpe` documentation

Huawei data storage

Deckhouse provides support for Huawei Dorado storage systems, enabling volume management in Kubernetes using a CSI driver through the creation of custom resources like HuaweiStorageClass. This solution ensures high-performance and fault-tolerant storage, making it an optimal choice for mission-critical workloads.

⚠ User-created StorageClass for the `csi.huawei.com` CSI driver is not allowed.
Only Huawei Dorado storage systems are supported. For other Huawei storage systems, contact the Deckhouse technical support.

This page provides instructions on connecting Huawei Dorado to Deckhouse, configuring the connection, creating StorageClass, and verifying storage functionality.

System requirements

- Presence of a deployed and configured Huawei storage system.
- Unique IQNs in `/etc/iscsi/initiatorname.iscsi` on each Kubernetes node.

Configuration

Note that all commands must be run on a machine that has administrator access to the Kubernetes API.

Enabling the module

To support Huawei Dorado storage systems, enable the `csi-huawei` module. This will ensure that all cluster nodes have:

- Registration of the CSI driver.
- Launch of service pods for the `csi-huawei` components.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-huawei
spec:
  enabled: true
  version: 1
EOF
```

Wait until the module transitions to the `Ready` state. Check the module's status with the following command:

```
d8 k get module csi-huawei -w
```

Creating a StorageClass

To create a StorageClass, you need to use the HuaweiStorageClass and HuaweiStorageConnection resource. Here is an example command to create such a resource:

- Creating a HuaweiStorageConnection resource:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: HuaweiStorageConnection
metadata:
  name: huaweistorageconn
spec:
  storageType: OceanStorSAN
  pools:
    - test
  urls:
    - https://192.168.128.101:8088
  login: "admin"
  password: "ivkerg43grdsf_"
  protocol: ISCSI
  portals:
    - 10.240.0.101
    - 10.250.0.101
  maxClientThreads: 30

EOF
```

Verify the creation of the object using the following command (`Phase` should be `Created`):

```
d8 k get huaweistorageconnections.storage.deckhouse.io <huaweistorageconnection name>
```

- Creating a HuaweiStorageClass resource:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: HuaweiStorageClass
metadata:
  name: huaweisc
spec:
  fsType: ext4
  pool: test
  reclaimPolicy: Delete
  storageConnectionName: huaweistorageconn
  volumeBindingMode: WaitForFirstConsumer
EOF
```

Verify the creation of the object using the following command (`Phase` should be `Created`):

```
d8 k get huaweistorageclasses.storage.deckhouse.io <huaweistorageclass name>
```

Module health verification

To verify module health, ensure that all pods in the `d8-csi-huawei` namespace are in the `Running` or `Completed` state and are running on every node in the cluster:

```
d8 k -n d8-csi-huawei get pod -owide -w
```

Additional resources

- Module `csi-huawei` documentation

NetApp data storage

Deckhouse Kubernetes Platform (DKP) implements support for NetApp data storage systems for volume management in Kubernetes using CSI driver. This integration provides reliable, scalable, and high-performance storage suitable for critical workloads. For working with NetApp storage systems, the `csi-netapp` module is used, which allows creating StorageClass in Kubernetes through creating a

NetappStorageClass resource.

⚠ Creating StorageClass for `csi.netapp.com` CSI driver by users is prohibited. Currently, the module supports storage systems compatible with NetApp Trident CSI. For support of other NetApp storage systems, please contact Deckhouse technical support.

This page provides instructions for connecting NetApp to DKP, configuring the connection, and creating StorageClass.

System Requirements

Before configuring work with the NetApp storage system, make sure that the following requirements are met:

- Availability of deployed and configured NetApp storage system.
- Unique IQNs in `/etc/iscsi/initiatorname.iscsi` on each Kubernetes node.

Configuring cluster integration with the NetApp storage system

To start working with the NetApp storage system, follow the step-by-step instructions below. Run all commands on a machine with administrative access to the Kubernetes API.

i To work with snapshots, the snapshot-controller module must be connected.

1. Execute the command to activate the `csi-netapp` module.

   ```
   d8 system module enable csi-netapp
   ```

   After activation, the following will be deployed on all cluster nodes:

   - CSI driver will be registered.
   - Service pods of `csi-netapp` components will be deployed.

2. Wait for the module to transition to the `Ready` state:

   ```
   d8 k get module csi-netapp -w
   ```

3. Ensure that all pods in the `d8-csi-netapp` namespace are in the `Running` or `Completed` state and deployed on all cluster nodes:

   ```
   d8 k -n d8-csi-netapp get pod -owide -w
   ```

4. Create a NetappStorageConnection resource:

   ```
   d8 k apply -f -<<EOF
   apiVersion: storage.deckhouse.io/v1alpha1
   kind: NetappStorageConnection
   metadata:
     name: netapp
   spec:
     controlPlane:
       address: "172.17.1.55"
       username: "admin"
       password: "password"
       svm: "svm1"
   EOF
   ```

5. Verify object creation with the following command (`Phase` should be `Created`):

   ```
   d8 k get netappstorageconnections.storage.deckhouse.io <netappstorageconnection name>
   ```

6. Create a NetappStorageClass resource:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: NetappStorageClass
metadata:
  name: netapp
spec:
  pool: "test-cpg"
  accessProtocol: "iscsi" # fc or iscsi (default iscsi), immutable parameter.
  fsType: "xfs" # xfs, ext3, ext4 (default ext4), configurable parameter.
  storageConnectionName: "netapp" # Immutable parameter.
  reclaimPolicy: Delete # Delete or Retain.
  cpg: "test-cpg"
EOF
```

7. Verify object creation with the following command (`Phase` should be `Created`):

```
d8 k get netappstorageclasses.storage.deckhouse.io <netappstorageclass name>
```

The NetApp storage system is now ready for use. You can use the created StorageClass to create PersistentVolumeClaims in your applications.

Additional resources

- Module `csi-netapp` documentation
- Module `snapshot-controller` documentation

NFS storage

Deckhouse Kubernetes Platform (DKP) supports integration with Network File System (NFS), providing the ability to use network file storage as Kubernetes volumes. The `csi-nfs` module provides a CSI driver for connecting to NFS servers and creating PersistentVolumes based on them.

This page provides instructions for configuring NFS storage in DKP, including connecting to an NFS server, creating StorageClass, configuring RPC-with-TLS security, and verifying system functionality.

System Requirements

Minimum Requirements

The following conditions must be met for the `csi-nfs` module to work:

- Supported Linux distributions with appropriate kernels.
- Configured and accessible NFS server.
- For RPC-with-TLS support: Linux kernel with enabled `CONFIG_TLS` and `CONFIG_NET_HANDSHAKE` options.

Configuration Recommendations

For optimal module operation, it is recommended to:

- For automatic pod restarts when TLS parameters change, enable the `pod-reloader` module (enabled by default).
- Use stable versions of NFS servers with support for required protocols.

> ⚠ For NFS to work as virtual disk storage in Deckhouse Virtualization Platform, configure the NFS server with the `no_root_squash` option.

Limitations

General Limitations

The following limitations apply when working with the `csi-nfs` module:

- Manual creation of StorageClass for CSI driver `nfs.csi.k8s.io` is prohibited — use the NFSStorageClass resource.

- PersistentVolumes are created only through NFSStorageClass resources.
- Changing NFS server parameters in already created PVs is impossible.

Volume Snapshot Limitations

> **i** A connected `snapshot-controller` module is required for working with snapshots.

Creating NFS volume snapshots has significant limitations related to the NFS architecture and the way they are implemented in `csi-nfs`. Avoid using snapshots in module whenever possible.

Snapshot Operation Principle:

- The CSI driver creates a snapshot at the NFS server level.
- The `tar` utility is used for archiving, which imposes certain limitations.
- The archive is saved in the root folder of the NFS server specified in the `spec.connection.share` parameter.

> **⚠**
> - Before creating a snapshot, **mandatory** stop the workload (pods) using the NFS volume.
> - NFS does not ensure atomicity of operations at the file system level when creating snapshots.

RPC-with-TLS Mode Limitations

Functional Limitations

The following limitations apply when using RPC-with-TLS:

- Only one client certificate is supported for the `mtls` security policy.
- One NFS server cannot simultaneously work in different security modes: `tls`, `mtls` and standard mode (without TLS).
- The `tlshd` daemon should not be running on cluster nodes, otherwise it will conflict with the module daemon. To prevent conflicts when enabling TLS on nodes, third-party `tlshd` is automatically stopped and its autostart is disabled.

System Requirements

The following system requirements must be met for RPC-with-TLS operation:

- Linux kernel must be compiled with enabled `CONFIG_TLS` and `CONFIG_NET_HANDSHAKE` parameters.
- `nfs-utils` package (in Debian-based distributions — `nfs-common`) version >= 2.6.3.

Configuration

> **i** All commands must be executed on a machine with administrative rights in the Kubernetes API.

The following steps are required to configure NFS storage:

- Module enabling.
- Creating NFSStorageClass.

Enabling the Module

To support working with NFS storage, enable the `csi-nfs` module, which allows creating StorageClass in Kubernetes using custom NFSStorageClass resources. After enabling the module, the following will happen on cluster nodes:

- CSI driver registration.
- Launch of `csi-nfs` service pods and creation of necessary components.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-nfs
spec:
  enabled: true
  version: 1
EOF
```

Wait for the module to transition to the `Ready` state:

```
d8 k get module csi-nfs -w
```

Check the status of pods in the `d8-csi-nfs` namespace. All pods should be in `Running` or `Completed` state and running on all nodes:

```
d8 k -n d8-csi-nfs get pod -owide -w
```

Creating StorageClass

To create a StorageClass, you must use the NFSStorageClass resource. Example of creating a resource:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: NFSStorageClass
metadata:
  name: nfs-storage-class
spec:
  connection:
    host: 10.223.187.3
    share: /
    nfsVersion: "4.1"
  reclaimPolicy: Delete
  volumeBindingMode: WaitForFirstConsumer
EOF
```

For each PV, a directory `<share directory>/<PV name>` will be created.

Volume Cleanup Configuration

When deleting a PersistentVolume, files with user data may remain on the volume. To ensure data security, you can configure the volume cleanup method before deletion using the `volumeCleanup` parameter.

⚠ Important notes on volume cleanup:

- The cleanup option does not affect files already deleted by the client application.
- Cleanup is performed only through the NFS protocol and depends on:
  - NFS server service.
  - File system.
  - Block device level and their virtualization (e.g., LVM).
  - Physical devices.
- Ensure NFS server trustworthiness before sending sensitive data.

Parameter `RandomFillSinglePass`

File contents are overwritten with a random sequence before deletion. The random sequence is transmitted over the network.

Parameter `RandomFillThreePass`

File contents are overwritten three times with random sequences before deletion. Three random sequences are transmitted over the network.

> **i** Using this method makes sense only if the server stores data on a hard disk, and there is a risk of physical access by an attacker to the device.

Parameter `Discard`

An optimized cleanup method that uses file system capabilities for working with solid-state drives. File contents are marked as free through the `falloc` system call with the `FALLOC_FL_PUNCH_HOLE` flag. The file system will free blocks fully used by the file through the `blkdiscard` call, and the remaining space will be overwritten with zeros.

Advantages of the `Discard` method:

- Traffic volume does not depend on file size, only on their quantity.
- Can ensure inaccessibility of old data in some server configurations.
- Works for both hard drives and solid-state drives.
- Allows increasing solid-state drive lifespan.

Changing NFS Server Parameters for Created PersistentVolumes

Changing NFS server parameters for created PersistentVolumes is impossible, as connection data to the NFS server is stored directly in the PersistentVolume manifest and cannot be changed. Changing NFSStorageClass will also not affect connection settings in existing PVs.

Creating Volume Snapshots

In `csi-nfs`, snapshots are created by archiving the volume folder. The archive is saved in the root of the NFS server folder specified in the `spec.connection.share` parameter. When creating volume snapshots, it’s important to consider the requirements and limitations listed above.

To create a snapshot, perform the following steps:

1. Enable the `snapshot-controller` module:

   ```
   d8 k apply -f - <<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: snapshot-controller
   spec:
     enabled: true
     version: 1
   EOF
   ```

2. Create a volume snapshot, specifying the required parameters:

   ```
   d8 k apply -f - <<EOF
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshot
   metadata:
     name: my-snapshot
     namespace: <namespace name where the PVC is located>
   spec:
     volumeSnapshotClassName: csi-nfs-snapshot-class
     source:
       persistentVolumeClaimName: <PVC name for which the snapshot needs to be
   created>
   EOF
   ```

3. Check the status of the created snapshot:

   ```
   d8 k get volumesnapshot
   ```

Issues with Deleting PVs with RPC-with-TLS Support

If the NFSStorageClass resource was configured with RPC-with-TLS support, a situation may arise where the PersistentVolume cannot be deleted. This happens due to secret deletion (e.g., after deleting NFSStorageClass) that stores mounting parameters. As a result, the controller cannot mount the NFS folder to delete the `<PV name>` folder.

Adding Multiple CAs to the `tlsParameters.ca` Parameter

To add multiple CA certificates to the `tlsParameters.ca` parameter, use the following commands:

For two CAs:

```
cat CA1.crt CA2.crt | base64 -w0
```

For three CAs:

```
cat CA1.crt CA2.crt CA3.crt | base64 -w0
```

For more CAs:

```
cat CA1.crt CA2.crt CA3.crt ... CAN.crt | base64 -w0
```

Additional resources

- Module `csi-nfs` documentation
- Module `pod-reloader` documentation
- Module `snapshot-controller` documentation

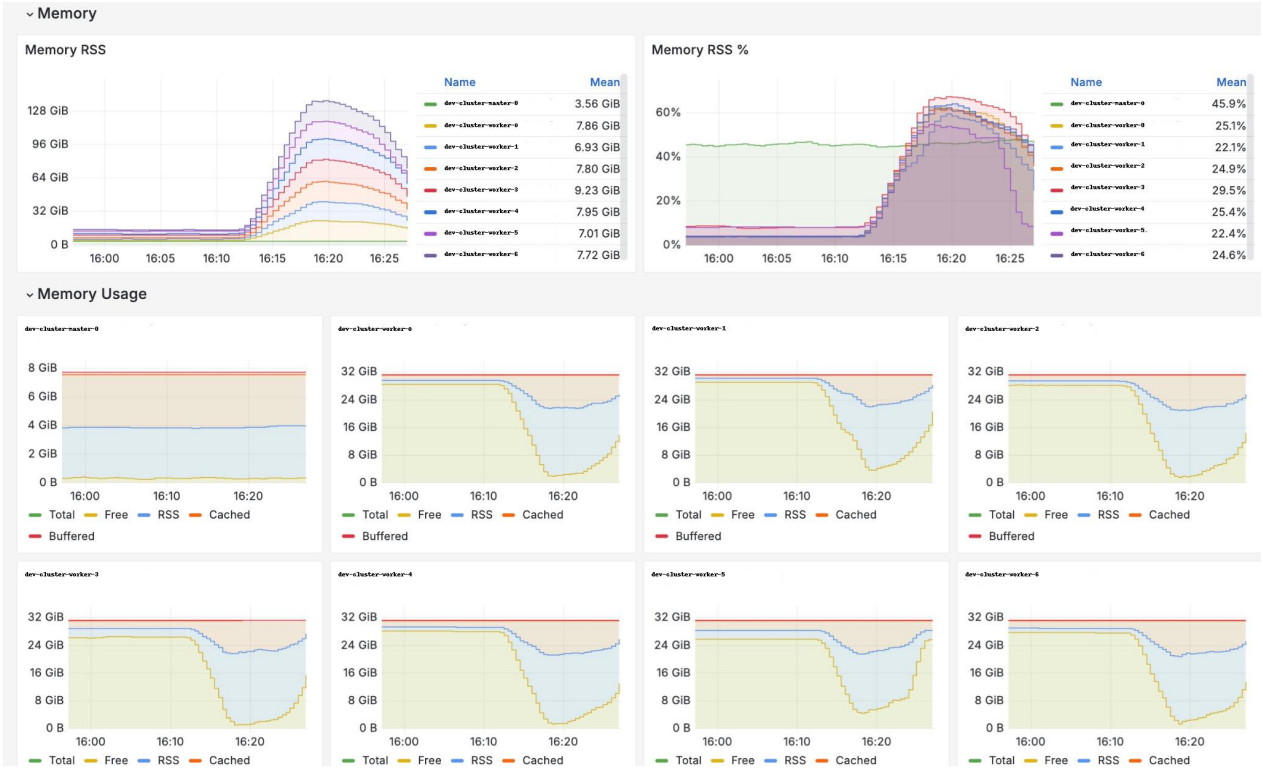
S3 object storage

Deckhouse supports working with S3-based object storage, enabling its use in Kubernetes for storing data as volumes. The GeeseFS file system is used, running via FUSE on top of S3, allowing S3 storage to be mounted as standard file systems.

This page provides instructions for setting up S3 storage in Deckhouse, including connection, creating a StorageClass, and verifying system functionality.

System requirements

- Kubernetes version 1.17+ with support for privileged containers.
- A configured S3 storage with available access keys.
- Sufficient memory on nodes. `GeeseFS` uses caching for working with files retrieved from S3. The cache size is set using the `maxCacheSize` parameter in S3StorageClass.
  Stress test results: 7 nodes, 600 Pods and PVCs, `maxCacheSize` = 500 MB, each Pod writes 300 MB, reads it, and then terminates.



Configuration

Note that all commands must be executed on a machine with administrative privileges in the Kubernetes API.

Required steps:

- Enable the module.
- Create S3StorageClass.

Enabling the module

To support working with S3 storage, enable the `csi-s3` module, which allows the creation of StorageClass and Secret in Kubernetes using custom resources like S3StorageClass.
After enabling the module, the following will occur on cluster nodes:

- CSI driver registration.
- Launch of `csi-s3` service Pods and creation of necessary components.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-s3
spec:
  enabled: true
  version: 1
EOF
```

Wait for the module to transition to the `Ready` state.

```
d8 k get module csi-s3 -w
```

Creating a StorageClass

The module is configured via the S3StorageClass manifest. Below is an example configuration:

```
apiVersion: storage.deckhouse.io/v1alpha1
kind: S3StorageClass
metadata:
  name: example-s3
spec:
  bucketName: example-bucket
  endpoint: https://s3.example.com
  region: us-east-1
  accessKey: <your-access-key>
  secretKey: <your-secret-key>
  maxCacheSize: 500
  insecure: false
```

If `bucketName` is empty then bucket in S3 will be created for each PV. If `bucketName` is not empty then folder inside the bucket will be created for each PV. If the specifies bucket does not exist it will be created.

Module health check

To verify the health of the module, ensure that all Pods in the `d8-csi-s3` namespace are in the `Running` or `Completed` state and are running on every node in the cluster:

```
d8 k -n d8-csi-s3 get pod -owide -w
```

Limitations

S3 is not a traditional file system, so it comes with several limitations. POSIX compatibility depends on the mounting module used and the specific S3 provider. Some storage backends may not guarantee data consistency (more details here).

Pay attention to the POSIX compatibility matrix.

Key limitations:

- File permissions, symbolic links, user-defined `mtimes`, and special files (block/character devices, named pipes, UNIX sockets) are not supported.
- Special file support is enabled by default for `Yandex S3` but disabled for other providers.
- File permissions are disabled by default.
- User-defined modification times are also disabled: `ctime`, `atime`, and `mtime` are always the same.
- The file modification time cannot be set manually (e.g., using `cp --preserve`, `rsync -a`, or `utimes(2)`).
- Hard links are not supported.
- File locking is not supported.
- “Invisible” deleted files are not retained. If an application keeps an open file descriptor after deleting a file, it will receive `ENOENT` errors when trying to access it.
- The default file size limit is 1.03 TB, achieved through chunking: 1000 parts of 5 MB, 1000 parts of 25 MB, and 8000 parts of 125 MB. The chunk size can be adjusted, but AWS enforces a maximum file size of 5 TB.

Known bugs

- The requested PVC volume size does not affect the created S3 bucket.
- `df -h` always reports the mounted storage size as 1 PB, and `used` does not change during usage.
- The CSI driver does not validate storage access credentials. Even with incorrect keys, the Pod will remain in `Running` status, and PersistentVolume and PersistentVolumeClaim will be `Bound`. Any attempt to access the mounted directory within the Pod will result in the Pod restarting.

Changing connection parameters for existing PVs

Connection parameters for storage cannot be changed. Changing the StorageClass does not update connection settings in existing PVs.

Displaying the mounted directory size

The `df -h` command shows the mounted volume size as 1 PB, and the `Used` value remains unchanged during operation. This is a peculiarity of the `GeeseFS` mounting module.

Handling exceeded bucket or user quotas

Exceeding the quota is an abnormal situation that should be avoided. The behavior in this case depends on the backend:

- Files can be copied and edited in Pods, but changes will not be reflected in the storage.
- The Pod may fail and restart.

Obtaining information about used space

To get information about the used space, use the storage web interface or a command-line utility.

Using multiple S3 storages in a single module

To do this, create an additional S3StorageClass and a corresponding PVC, then mount them in a Pod as follows:

```
d8 k apply -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: csi-s3-test-nginx
  namespace: default
spec:
  containers:
    - name: csi-s3-test-nginx
      image: nginx
      volumeMounts:
        - mountPath: /usr/share/nginx/html/s3
          name: webroot
        - mountPath: /opt/homedir
          name: homedir
  volumes:
    - name: webroot
      persistentVolumeClaim:
        claimName: csi-s3-pvc # PVC name
        readOnly: false
    - name: homedir
      persistentVolumeClaim:
        claimName: csi-s3-pvc2 # PVC-2 name
        readOnly: false
  EOF
```

Using a single bucket for multiple Pods

When the `bucketName` parameter is specified in S3StorageClass, separate directories are created for each PV in the same bucket, allowing one bucket to be used for multiple Pods.

Troubleshooting

Issues creating a PVC

If you encounter problems when creating a PVC, check the `provisioner` logs:

```
d8 k -n d8-csi-s3 logs -l app=csi-provisioner-s3 -c csi-s3
```

Issues starting containers

Make sure that MountPropagation is not set to `false`, and check the S3 driver logs:

```
d8 k -n d8-csi-s3 logs -l app=csi-s3 -c csi-s3
```

Additional resources

- Module `csi-s3` documentation

SCSI-based data storage

Deckhouse Kubernetes Platform (DKP) supports managing storage connected via iSCSI or Fibre Channel, enabling working with volumes at the block device level. This allows for the integration of storage systems with Kubernetes and management through a CSI driver.

This page provides instructions for connecting SCSI devices in DKP, creating SCSITarget, StorageClass, and verifying system functionality.

Supported features

DKP supports:

- Detecting Logical Unit Numbers (LUN) via iSCSI or Fibre Channel.
- Creating PersistentVolume (PV) from pre-provisioned LUN.
- Deleting PersistentVolume and wiping data on LUN.
- Attaching LUN to nodes via iSCSI or Fibre Channel.
- Creating multipath devices and mounting them in pods.
- Detaching LUN from nodes.

Limitations

DKP doesn’t support:

- Creating LUN on storage systems.
- Resizing LUN.
- Creating snapshots.

System requirements

The following requirements are applicable to the infrastructure and cluster nodes:

- A deployed and configured storage system providing access to LUN via iSCSI or Fibre Channel.
- For iSCSI connections:
  - A unique iSCSI Qualified Name (IQN) must be configured on each cluster node in the `/etc/iscsi/initiatorname.iscsi` file.
  - The `multipath-tools` package must be installed on nodes.
- For Fibre Channel connections:
  - Fibre Channel Host Bus Adapters (FC HBA) must be installed and available on cluster nodes (`/sys/class/fc_host/host*`).
  - In the Storage Area Network (SAN), LUN zoning and masking must be configured, providing node initiators with access to the storage target ports.
  - Necessary LUNs must be pre-provisioned on the storage system. DKP does not create LUNs.
  - The `multipath-tools` package must be installed on nodes.

Quick start

All commands should be executed on a machine with access to the Kubernetes API and administrator rights.

Enabling the csi-scsi-generic module

Enable the `csi-scsi-generic` module using the command below. This will ensure that the following happens on all cluster nodes:

- The CSI driver is registered.
- Auxiliary pods for the `csi-scsi-generic` components are running.

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-scsi-generic
spec:
  enabled: true
  version: 1
EOF
```

Wait for the module to transition to the `Ready` state.

```
d8 k get module csi-scsi-generic -w
```

Creating an SCSITarget

The SCSITarget resource describes connection to a single SCSI target. When creating a resource in `spec`, specify one of the connection types: `iSCSI` or `fibreChannel`.

iSCSI

The following is a configuration example for connecting via iSCSI. In this example, two SCSITarget resources are created.

You can create multiple resources for either the same or different storage systems. This lets you use multipath to improve failover and performance.

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: SCSITarget
metadata:
  name: hpe-3par-1
spec:
  deviceTemplate:
    metadata:
      labels:
        my-key: some-label-value
  iSCSI:
    auth:
      login: ""
      password: ""
    iqn: iqn.2000-05.com.3pardata:xxxx1
    portals:
    - 192.168.1.1

---
apiVersion: storage.deckhouse.io/v1alpha1
kind: SCSITarget
metadata:
  name: hpe-3par-2
spec:
  deviceTemplate:
    metadata:
      labels:
        my-key: some-label-value
  iSCSI:
    auth:
      login: ""
      password: ""
    iqn: iqn.2000-05.com.3pardata:xxxx2
    portals:
    - 192.168.1.2
EOF
```

To verify that the object has been created, run:

```
d8 k get scsitargets.storage.deckhouse.io <name-of-scsitarget>
```

An object is considered to be created successfully if it says `Created` in the corresponding `Phase` column in the output.

Fibre Channel

To configure a connection based on Fibre Channel, do the following:

1. Before creating the SCSITarget resource, configure zoning and LUN masking on the SAN so that cluster nodes can access the necessary LUNs.
2. Create the SCSITarget resource. In the `spec.fibreChannel.WWNs` field, define a World Wide Port Name (WWPN) of the target storage ports, which expose the necessary LUNs. DKP discovers devices by matching the defined WWPNs against paths under `/dev/disk/by-path/`.

   Define the WWPNs in one of the following formats:

   - 16 hexadecimal characters (`2001c89f1acd6117`)
   - With colons (`20:01:c8:9f:1a:cd:61:17`)
   - With the `0x` prefix (`0x2001c89f1acd6117`)

   DKP triggers an FC host scan during discovery and volume attach. It does not configure the commutators or log in to targets.

Configuration example with two target storage system ports for multipath:

```shell kubectl apply -f -«EOF apiVersion: storage.deckhouse.io/v1alpha1 kind: SCSITarget metadata: name: hpe-3par-fc-1 spec: deviceTemplate: metadata: labels: my-key: some-label-value fibreChannel: WWNs: - 2001c89f1acd6117

---

apiVersion: storage.deckhouse.io/v1alpha1 kind: SCSITarget metadata: name: hpe-3par-fc-2 spec: deviceTemplate: metadata: labels: my-key: some-label-value fibreChannel: WWNs: - 2001c89f1acd6118 EOF
```

1. Before applying the resource verify that FC HBA are available on the node and that LUN paths are visible after SAN zoning:

   ```shell
   # FC hosts present.
   ls /sys/class/fc_host/

   # Target WWPNs and LUNs available after zoning.
   ls -l /dev/disk/by-path/ | grep -E 'fc-|/fc-'
   ```

Possible results:

- If the `/sys/class/fc_host/` directory is empty, the node has no FC HBA or the corresponding driver is not loaded.
- If there are no `fc-*` entries in the `/dev/disk/by-path/` directory, check zoning and LUN masking settings on the storage system.
- If the FC paths are displayed, the node is ready to detect LUN provided via Fibre Channel.

1. After the SCSITarget is created, the controller discovers available LUNs and creates SCSIDevice objects. Use the same SCSIStorageClass resource and `scsiDeviceSelector` workflow as for iSCSI.

2. To verify that the object has been created, run the following command:

   ```
   d8 k get scsitargets.storage.deckhouse.io <name-of-scsitarget>
   ```

   An object is considered to be created successfully if it says `Created` in the corresponding `Phase` column in the output.

Creating a StorageClass

To create a StorageClass, use the SCSIStorageClass resource. A command example to create such a resource:

```
d8 k apply -f -<<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: SCSIStorageClass
metadata:
  name: scsi-all
spec:
  scsiDeviceSelector:
    matchLabels:
      my-key: some-label-value
  reclaimPolicy: Delete
EOF
```

The `scsiDeviceSelector` field is used to select the SCSITarget for PV creation based on labels. In the example above, all SCSITargets with the label `my-key: some-label-value` are selected. This label will be applied to all devices detected within the specified SCSITarget.

To verify that the object has been created, run:

```
d8 k get scsistorageclasses.storage.deckhouse.io <name-of-scsistorageclass>
```

An object is considered to be created successfully if it says `Created` in the corresponding `Phase` column in the output.

Csi-scsi-generic module health check

Check the status of pods in the `d8-csi-scsi-generic` namespace using the command below. All pods should be in the `Running` or `Completed` state and deployed on all nodes.

```
d8 k -n d8-csi-scsi-generic get pod -owide -w
```

Additional resources

- Module `csi-scsi-generic` documentation

TATLIN.UNIFIED (Yadro) unified storage

Deckhouse supports integration with the TATLIN.UNIFIED (Yadro) storage system, enabling volume management in Kubernetes. This allows the use of centralized storage for containerized workloads, ensuring high performance and fault tolerance.

This page provides instructions on connecting TATLIN.UNIFIED (Yadro) to Deckhouse, configuring the connection, creating a StorageClass, and verifying system functionality.

Enabling the module

To manage volumes based on the TATLIN.UNIFIED (Yadro) storage system in Deckhouse, the `csi-yadro-tatlin-unified` module is used. It allows the creation of StorageClass resources through custom resources like YadroTatlinUnifiedStorageClass. To enable the module, run the following command:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-yadro-tatlin-unified
spec:
  enabled: true
  version: 1
EOF
```

Wait until `csi-yadro-tatlin-unified` is in the `Ready` status. To check the status, run the following command:

```
d8 k get module csi-yadro-tatlin-unified -w
```

In the output, you should see information about the module:

```
NAME                        STAGE   SOURCE    PHASE       ENABLED    READY
si-yadro-tatlin-unified             Embedded  Available   True       True
```

Connecting to the TATLIN.UNIFIED storage system

To connect to the `TATLIN.UNIFIED` storage system and enable configuring of StorageClass objects, apply the following YadroTatlinUnifiedStorageConnection resource:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: YadroTatlinUnifiedStorageConnection
metadata:
  name: yad1
spec:
  controlPlane:
    address: "172.19.28.184"
    username: "admin"
    password: "cGFzc3dvcmQ=" # Must be encoded in Base64.
    ca: "base64encoded"
    skipCertificateValidation: true
  dataPlane:
    protocol: "iscsi"
    iscsi:
      volumeExportPort: "p50,p51,p60,p61"
EOF
```

Creating a StorageClass

To create a StorageClass, use the YadroTatlinUnifiedStorageClass resource. Creating a StorageClass resource manually without using YadroTatlinUnifiedStorageClass can lead to errors.

Example command to create a StorageClass based on the `TATLIN.UNIFIED` storage system:

```
d8 k apply -f - <<EOF
apiVersion: storage.deckhouse.io/v1alpha1
kind: YadroTatlinUnifiedStorageClass
metadata:
  name: yad1
spec:
  fsType: "xfs"
  pool: "pool-hdd"
  storageConnectionName: "yad1"
  reclaimPolicy: Delete
EOF
```

Ensuring the module works

To make sure the `csi-yadro-tatlin-unified` module is working properly, check the pod status in the `d8-csi-yadro-tatlin-unified` namespace. All pods must have the `Running` or `Completed` status. The `csi-yadro-tatlin-unified` pods must be running on all nodes.

To check that the module works, run the following command:

```
d8 k -n d8-csi-yadro-tatlin-unified get pod -owide -w
```

Additional resources

- Module `csi-yadro-tatlin-unified` documentation

Security

The “Security” section covers security features in Deckhouse Kubernetes Platform. It contains recommendations, instructions, and configuration examples for built-in protection mechanisms, as well as integration with external systems.

In this section, you will find information on:

- Security event audit:
  - How to enable and configure Kubernetes API event audit.
  - How to collect security events at the kernel and Kubernetes API levels using the platform’s built-in capabilities (Falco).
  - How to configure audit rules and receive alerts on suspicious activity.

- Security policies:
  - Support for Pod Security Standards.
  - Configuring operational and advanced security policies using Gatekeeper.
  - Verifying container image signatures.
  - Working with custom policies and exceptions.
- Image vulnerability scanning:
  - How to set up regular scanning of container images.
  - How to view scan results and manually trigger rescans.
- Certificate management:
  - Issuing, renewing, and managing TLS certificates using the `cert-manager` module.
  - Examples of using Let’s Encrypt, HashiCorp Vault, self-signed, and external CAs.
  - Support for `HTTP-01` and `DNS-01` validation types.
- Integration with external monitoring and security systems:
  - Sending logs to Kaspersky Unified Monitoring and Analysis Platform (KUMA).
  - Configuring exclusions for antivirus solutions, using Kaspersky Endpoint Security for Linux (KESL) as an example.

Additional resources

- Module `cert-manager` documentation

Security events

Deckhouse Kubernetes Platform (DKP) provides tools for declarative collection, processing, normalization, and delivery of security events extracted from logs of applications and Kubernetes infrastructure components.

A security event is a structured record of an action or fact that is significant from an information security perspective.

DKP lets you:

- Collect security events from various sources (pod logs, node files, Kubernetes API audit).
- Bring events to a unified format with a mandatory minimum set of attributes.
- Enrich events with contextual data.
- Filter events by sources and severity level.
- Deliver events to storage and analytics systems (Loki, Elasticsearch, Kafka, Splunk, Vector, and others).

The `security-events-manager` module is responsible for collecting, processing, and delivering security events. The auxiliary `log-shipper` module is used for log collection.

Dependencies and requirements

The `security-events-manager` module requires the following DKP modules:

- `log-shipper`: Collects logs from pod sources and node files, performs preliminary record selection.
- `loki`: Provides in-cluster storage for security events (used by default as a destination).

For details about the security events architecture, refer to “Security events architecture”.

Data sources for security events

The `security-events-manager` module collects data from two types of sources:

- **Container sources**: Application logs in Kubernetes pods. Collection is performed via the `log-shipper` module, which selects records by pod labels and namespaces.
- **Cluster sources**: Node files and system service logs not bound to a specific namespace. For example:
  - `/var/log/kube-audit/audit.log`: Kubernetes API audit log.
  - `/var/log/auth.log`: Node system authentication event log.

Collection is made in two steps:

1. The `log-shipper` module performs preliminary selection of log records using simple comparison operators (`In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`) and forwards them to the `security-events-manager` gateway.
2. The `security-events-manager` gateway parses the records, transforms them to a unified model, and delivers to a configured destination.

Since log parsing is resource-intensive, it is performed only for records pre-selected as potentially containing security events.

Enabling security events

To enable security events, follow these steps:

1. Enable the required modules if they are not yet enabled:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: log-shipper
spec:
  enabled: true
---
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: loki
spec:
  enabled: true
```

2. Enable the `security-events-manager` module:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: security-events-manager
spec:
  enabled: true
```

All available `security-events-manager` parameters are listed in the module documentation.

Configuring event sources

Log collection and processing are configured through the following custom resources:

- PodSecurityEventShipper: For container sources (pod logs in a specific namespace).
- ClusterSecurityEventShipper: For cluster sources (node files, system service logs).

These resources configure:

- Log source (`source` and `input` fields)
- Preliminary record selection rules (`produces` field)
- Parsing rules (`parser` or `parserRef` field with a reference to SecurityEventLoggingTransformationRules or ClusterSecurityEventLoggingTransformationRules)
- Transformation and enrichment rules (`transform` and `enrich` fields)

PodSecurityEventShipper configuration example defining a container source:

```
apiVersion: security.deckhouse.io/v1alpha1
kind: PodSecurityEventShipper
metadata:
  name: runtime-audit-engine
  namespace: d8-runtime-audit-engine
spec:
  - source: runtime-audit-engine
    input:
      type: KubernetesPods
      kubernetesPods:
        labelSelector:
          matchLabels:
            app: runtime-audit-engine
    produces:
      - eventCode: K8S_POD_CREATED
        extract:
          field: message
          operator: Regex
          values:
            - ".*K8s Pod Created.*"
```

ClusterSecurityEventShipper configuration example defining a cluster source:

```
apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventShipper
metadata:
  name: kube-audit
spec:
  - source: kube-audit
    input:
      type: File
      files:
        - /var/log/kube-audit/audit.log
    produces:
      - eventCode: UNAUTHORIZED_ACCESS
        extract:
          field: message
          operator: Regex
          values:
            - ".*\"code\":401.*"
```

Configuring event delivery

After events are formed, you should define the destinations they will be delivered to. For this, you need to:

1. Configure security event destinations.
2. Configure event delivery rules to destinations.

Configuring a destination

Destinations are configured via the ClusterSecurityEventDestination resource.

The `security-events-manager` module supports all destination types available in the `log-shipper` ecosystem (Loki, Elasticsearch, Kafka, Splunk, Vector, File, and others). For in-cluster security event storage, automatic Loki destination configuration is available when the corresponding module option is enabled.

ClusterSecurityEventDestination configuration example defining the Loki destination:

```
apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventDestination
metadata:
  name: cluster-loki
spec:
  type: Loki
  loki:
    auth:
      strategy: Bearer
      token: <EXAMPLE>
    endpoint: https://loki.d8-monitoring:3100
    tls:
      verifyCertificate: true
      verifyHostname: true
      ca: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t...
```

Configuring event delivery rules

When configuring security event delivery rules, you need to define:

- Sources that will be sending the events
- Minimum severity level for sending
- Target destinations

For this, use the ClusterSecurityEventConfig resource. It defines sources (in the format of exact names or masks), minimum severity level, and an array of destinations (resources of type ClusterSecurityEventDestination) that will receive the selected events.

ClusterSecurityEventConfig configuration example for sending events from the `kube-audit`, `runtime-audit-engine` and `user-authn` sources to the `cluster-loki` destination:

```
apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventConfig
metadata:
  name: default
spec:
  defaultSeverityThreshold: Low
  enabledSources:
    - clusterSecurityEventShipper/kube-audit/kube-audit
    - podSecurityEventShipper/d8-runtime-audit-engine/runtime-audit-engine/falco
    - podSecurityEventShipper/d8-user-authn/user-authn/dex
  # Alternatively, use masks.
  # enabledSourcesMasks:
  #   - clusterSecurityEventShipper/kube-audit/*
  #   - podSecurityEventShipper/*

  destinations:
    - cluster-loki
```

Default module settings

If the module settings are not explicitly specified, the following objects are created in the cluster:

- ClusterSecurityEventConfig: Configures security event delivery to destinations. By default, an object with the following configuration is created:

```
apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventConfig
metadata:
  name: default
spec:
  defaultSeverityThreshold: Medium
  destinations:
    - cluster-loki
  enabledSourcesMasks:
    - podSecurityEventShipper/*
    - clusterSecurityEventShipper/*
```

This object configuration is controlled by the module parameter `securityEventConfig`.

- ClusterSecurityEventDestination: Configures security event destination. By default, the following object is generated that allows sending events to the in-cluster Loki service:

```
apiVersion: security.deckhouse.io/v1alpha1
kind: ClusterSecurityEventDestination
metadata:
  name: cluster-loki
spec:
  type: Loki
  loki:
    auth:
      strategy: Bearer
      token: <token> # Filled automatically.
    endpoint: https://loki.d8-monitoring:3100
```

You can disable generation of the default destination via the dedicated parameter `clusterSecurityEventDestination.clusterLoki`.

Supported security events

The `security-events-manager` module ships with a built-in set of security event detection rules covering authentication, configuration, RBAC, runtime, and other categories. For the up-to-date list of implemented security events, their codes, severity levels, and descriptions, refer to the module documentation.

Additional resources

- Module `log-shipper` documentation
- Module `loki` documentation
- Module `security-events-manager` documentation

Kubernetes API event audit

The Kubernetes auditing feature allows you to track requests to the API server and analyze events occurring in the cluster. Auditing can be useful for troubleshooting unexpected behavior and for meeting security requirements.

Kubernetes supports audit configuration via the Audit policy mechanism, which allows you to define logging rules for target operations. By default, audit results are written to the `/var/log/kube-audit/audit.log` file.

Built-in audit policies

Deckhouse Kubernetes Platform (DKP) configures a default audit policy that logs the following:

General events:

- Creation and deletion of cluster nodes (`Node` objects);
- Create, update, and delete operations on resources in all system namespaces (`kube-system`, `d8-*`);
- Requests from service accounts in system namespaces (`kube-system`, `d8-*`);
- Bulk `LIST` requests to all resources (used to diagnose high API server resource consumption);

- Operations related to `virtualization` module resources;
- Actions in the `d8-virtualization` namespace;
- Operations on `ModuleConfig` objects;
- Requests from unauthenticated users (only `Metadata` level events are recorded).

Security-sensitive events:

- Creation, modification, and deletion of `Pod` objects;
- Use of container image references without a `@sha256` digest;
- Creation and deletion of `ServiceAccount` objects, including in system namespaces;
- Creation, modification, and deletion of `Role` and `ClusterRole` objects;
- Creation, modification, and deletion of `ClusterRoleBinding` objects;
- Use of `attach` and `exec` commands against Pods, as well as adding ephemeral containers (get and patch operations on `pods/exec`, `pods/attach`, and `pods/ephemeralcontainers`).

Events explicitly excluded from audit logging (because the corresponding objects change very frequently):

- Operations on `Endpoints`, `EndpointSlice`, and `Event` objects;
- Operations on `Lease` objects (raft leader election in platform components);
- Actions on `ConfigMap` objects used for raft leader election (`cert-manager-cainjector-leader-election`, `cert-manager-controller`, `ingress-nginx`, and similar);
- Operations on `VerticalPodAutoscalerCheckpoints` objects;
- `PATCH` operations on `VerticalPodAutoscaler` objects performed by the `d8-vertical-pod-autoscaler-recommender` service account;
- Actions on `UpmeterHookProbes` objects;
- Any operations in the `d8-upmeter` namespace.

These policies are enabled by default. To disable them, set the `basicAuditPolicyEnabled` parameter to `false`.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 1
  settings:
    apiserver:
      auditPolicyEnabled: true
      basicAuditPolicyEnabled: false
```

Configuring a custom audit policy

To create an advanced Kubernetes API audit policy, follow these steps:

1. Enable the `auditPolicyEnabled` parameter in the `control-plane-manager` module configuration:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: control-plane-manager
   spec:
     version: 1
     settings:
       apiserver:
         auditPolicyEnabled: true
   ```

2. Create the `kube-system/audit-policy` Secret containing the policy YAML file encoded in Base64:

```
apiVersion: v1
kind: Secret
metadata:
  name: audit-policy
  namespace: kube-system
data:
  audit-policy.yaml: <Base64>
```

Example `audit-policy.yaml` content with a minimal working configuration:

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: Metadata
  omitStages:
  - RequestReceived
```

For more information on possible contents of `audit-policy.yaml`, refer to the following sources:

- Kubernetes official documentation
- GCE helper script code snippet

How to generate the contents of the `audit-policy.yaml` file

Kubernetes Documentation Policy Resource Field Structure

A Kubernetes audit policy is defined in a YAML file and consists of a set of rules that determine which events and at what level of detail will be logged. The file has the following structure:

```
apiVersion: audit.k8s.io/v1  # API version used for the audit policy
kind: Policy              # Resource type — constant Policy
rules:                     # Set of rules for auditing
  - level:                 # Log detail level. Required field
    users:                 # List of users whose actions are logged
    userGroups:             # User groups (e.g., system:serviceaccounts)
    verbs:                 # Actions/operations that are logged (create, update,
delete, etc.)
    resources:              # Kubernetes resources for which the rule applies
    namespaces:             # Namespaces that the rule covers
```

The `rules` array describes the audit rules. Each rule contains the following fields:

- `level` — The detail level of the logged event.

  Possible values (from most to least detailed):

  - `None`: Do not log at all.
  - `Metadata`: Only request metadata (who, when, what, where; without object contents).
  - `Request`: Also stores the request body (for update requests only).
  - `RequestResponse`: Stores both the request body and the response content.

- `users`: A list of usernames the rule applies to (e.g., `["admin"]`). For service accounts, the name typically looks like `system:serviceaccount:<namespace>:<serviceaccount-name>`. For regular users, the name depends on the authentication system settings. For `deckhouse.io/v1` `Users` objects, `email` is used as the name.

- `userGroups`: User groups (e.g., `["system:authenticated"]`). After authentication, kube-apiserver assigns a list of groups to each user (e.g., all authenticated users are part of `system:authenticated`, service accounts are in additional groups). If a request comes from a user who is a member of at least one of the groups specified in userGroups, the rule is applied to that request.

  Kubernetes built-in groups:

  - `system:authenticated`: Everyone who has been authenticated.
  - `system:unauthenticated`: Requests from anonymous users.
  - `system:serviceaccounts`: All service accounts from all namespaces.

    - `system:serviceaccounts:<namespace>`: Service accounts in a specific namespace.
- `verbs`: List of API operations (get, list, create, delete, etc.).
- `resources`: Array of target resources:
    - `group`: API group (e.g., `"apps"`, `"batch"`, `""` for core).
    - `resources`: Resource types (e.g., `"[pods", "deployments"]`). A full list of resources and their groups can be obtained using the `kubectl api-resources` command
- `namespaces`: Array of namespaces where the rule applies.
- `nonResourceURLs`: A set of URL paths to audit. The * character is allowed, but only as a full, final step of the path. Examples:
    - `/metrics` — log requests to apiserver metrics
    - `/healthz*` — log all health requests

Examples

Log all requests from all authenticated users

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  - level: Metadata
    userGroups: ["system:authenticated"]
```

⚠ **Warning!** Not recommended for production environments. A high event flow may result in increased load on the control-plane disk subsystem of cluster nodes.

Example log entry:

```
{
  "stage": "ResponseComplete",
  "requestURI": "/healthz",
  "verb": "get",
  "user": {
    "username": "kube-apiserver-kubelet-client",
    "groups": [
      "kubeadm:cluster-admins",
      "system:authenticated"    //  <- everyone who has been authenticated
    ]
  },
  ...
}
```

Logging all requests not from service accounts

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  - level: Metadata
    userGroups: ["system:authenticated"]
    users: []
    omitStages: []
    # Exclude service accounts via a deny rule
  - level: None
    userGroups: ["system:serviceaccounts"]
```

Example log entry:

```
{
  "user": {
    "username": "user@example.com",
    "groups": ["admins","system:authenticated"]
  },
  ...
}
```

Logging all operations with standard resources

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
  - level: Metadata
    resources:
      - group: "apps"
        resources: ["deployments", "daemonsets", "statefulsets", "replicasets"]
      - group: "batch"
        resources: ["jobs", "cronjobs"]
      - group: ""
        resources: ["*"]
```

Working with the audit log file

On DKP master nodes, it is assumed that a log collection tool (`log-shipper`, `promtail`, or `filebeat`) is installed to monitor the `/var/log/kube-audit/audit.log` file.

The log rotation settings for this file are predefined and cannot be changed:

- Maximum file size: 1000 MB.
- Maximum retention period: 30 days.

Depending on the policy configuration and the volume of requests to the API server, the number of log entries can be very large. In such cases, the retention period may be reduced to less than 30 minutes.

⚠ Unsupported options or typos in the configuration file may cause the API server to fail to start.

If the API server fails to start, take the following steps:

1. Manually remove the `--audit-log-*` parameters from the `/etc/kubernetes/manifests/kube-apiserver.yaml` manifest.
2. Restart the API server with the following command:

   ```
   docker stop $(docker ps | grep kube-apiserver- | awk '{print $1}')

   # Alternative option (depending on the CRI in use).
   crictl stopp $(crictl pods --name=kube-apiserver -q)
   ```

3. After restarting, fix the Secret or delete it with the following command:

   ```
   d8 k -n kube-system delete secret audit-policy
   ```

Redirecting the audit log file to stdout

By default, the audit log is saved to the `/var/log/kube-audit/audit.log` file on master nodes. If necessary, you can redirect its output to the `kube-apiserver` process stdout instead of a file by setting the `apiserver.auditLog.output` parameter in the `control-plane-manager` module to `Stdout`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 1
  settings:
    apiserver:
      auditPolicyEnabled: true
      auditLog:
        output: Stdout
```

In this case, the log will be available in the `kube-apiserver` container stdout.

Then, using the built-in DKP logging mechanism, you can configure log collection and forwarding to your own security system.

Additional resources

- Module `control-plane-manager` documentation

Runtime audit

Deckhouse Kubernetes Platform (DKP) provides built-in tools for detecting security threats by analyzing Linux kernel events and auditing Kubernetes API events. With DKP, you can:

- Detect threats in environments by analyzing applications and containers.
- Identify attempts to exploit vulnerabilities from the CVE database and signs of cryptocurrency miner activity.
- Detect Kubernetes-specific threats, including:
  - Command-line shells launched in containers or pods.
  - Containers running in privileged mode.
  - Mounting insecure paths into containers (for example, `/proc`).
  - Attempts to read sensitive data (for example, from `/etc/shadow`).

Data sources for runtime audit

DKP uses two main sources of events:

- Linux kernel events — via the eBPF driver for the Falco threat detection system.
- Kubernetes API audit events — via integration with Kubernetes auditing and a webhook interface.

For details about the runtime audit architecture, refer to “Runtime audit architecture”.

Minimum requirements

To capture kernel events, you need:

- Linux kernel version 5.8 or higher.
- Support for eBPF. You can check for support in one of the following ways:
  - Make sure the `/sys/kernel/btf/vmlinux` file exists:

    ```
    ls -lah /sys/kernel/btf/vmlinux
    ```

  - Make sure the `CONFIG_DEBUG_INFO_BTF` option is enabled:

    ```
    grep -E "CONFIG_DEBUG_INFO_BTF=(y|m)" /boot/config-*
    ```

Falco agents run on each cluster node and consume resources depending on the number of rules applied and events collected.

> **i** On some systems, eBPF probes may not work.

Enabling runtime audit

To enable the runtime audit, follow these steps:

1. Make sure the nodes meet the minimum requirements.
2. Enable the `runtime-audit-engine` module using the following configuration:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: runtime-audit-engine
   spec:
     enabled: true
   ```

3. (**Optional**) If the cluster control plane is not managed by DKP with `control-plane-manager`, configure the Kubernetes API audit webhook manually.

All available runtime audit parameters are listed in the `runtime-audit-engine` module documentation.

Manually configuring the Kubernetes API audit webhook

> **i** Webhook configuration is not required if the `control-plane-manager` module is enabled. In this case, when the `runtime-audit-engine` module is enabled, the settings for collecting Kubernetes API audit events will be applied automatically.

To configure the webhook for receiving audit events from `kube-apiserver`:

1. Create a `kubeconfig` file for the webhook with the address `https://127.0.0.1:9765/k8s-audit` and certificate data (`ca.crt`) from the `d8-runtime-audit-engine/runtime-audit-engine-webhook-tls` Secret.

   Example:

   ```
   apiVersion: v1
   kind: Config
   clusters:
     - name: webhook
       cluster:
         certificate-authority-data: BASE64_CA
         server: "https://127.0.0.1:9765/k8s-audit"
   users:
     - name: webhook
   contexts:
     - context:
         cluster: webhook
         user: webhook
       name: webhook
   current-context: webhook
   ```

2. Specify the path to the configuration file with the `--audit-webhook-config-file` flag in the `kube-apiserver` manifest.
3. (**Optional**) To collect Kubernetes API audit events from not only system but also user namespaces, configure audit policies.

Working with audit rules

Runtime audit event analysis is based on rules that define suspicious behavior criteria. DKP includes:

- **Built-in rules**, including:
  - Kubernetes audit rules (located in the `falco` container at `/etc/falco/k8s_audit_rules.yaml`).

  To configure the list of built-in rules, use the `settings.builtInRulesList` parameter of the `runtime-audit-engine` module.
- **Custom rules**, defined via the FalcoAuditRules custom resource.

For more information on how runtime audit rules work, refer to “Runtime audit architecture”.

Adding a custom rule

To add a rule, create a FalcoAuditRules resource with the required conditions. Use the Falco condition syntax. Falco agents will automatically apply the created rule.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: FalcoAuditRules
metadata:
  name: ownership-permissions
spec:
  rules:
    - macro:
        name: spawned_process
        condition: (evt.type in (execve, execveat) and evt.dir=<)
    - rule:
        name: Detect Ownership Change
        desc: detect file permission/ownership change
        condition: >
          spawned_process and proc.name in (chmod, chown) and proc.args contains
"/tmp/"
        output: >
          The file or directory below has had its permissions or ownership changed
(user=%user.name
          command=%proc.cmdline file=%fd.name parent=%proc.pname
pcmdline=%proc.pcmdline gparent=%proc.aname[2])
        priority: Warning
        tags: [filesystem]
```

For more rule examples, see:

- Falco rules official repository
- Falco rules on Artifact Hub

Applying a third-party rule

Since Falco rule structure differs from DKP custom resource format, third-party rules from the internet must be converted to a FalcoAuditRules resource before applying them.

Use the following script to convert:

```
git clone github.com/deckhouse/deckhouse
cd deckhouse/ee/modules/runtime-audit-engine/hack/far-converter
go run main.go -input /path/to/falco/rule_example.yaml > ./my-rules-cr.yaml
```

Example conversion result:

- Original rule:

```
# /path/to/falco/rule_example.yaml
- macro: spawned_process
   condition: (evt.type in (execve, execveat) and evt.dir=<)

- rule: Linux Cgroup Container Escape Vulnerability (CVE-2022-0492)
   desc: "This rule detects an attempt to exploit a container escape
vulnerability in the Linux Kernel."
   condition: container.id != "" and proc.name = "unshare" and spawned_process
and evt.args contains "mount" and evt.args contains "-o rdma" and evt.args
contains "/release_agent"
   output: "Detect Linux Cgroup Container Escape Vulnerability (CVE-2022-0492)
(user=%user.loginname uid=%user.loginuid command=%proc.cmdline
args=%proc.args)"
   priority: CRITICAL
   tags: [process, mitre_privilege_escalation]
```

- Converted resource:

```
# ./my-rules-cr.yaml
apiVersion: deckhouse.io/v1alpha1
kind: FalcoAuditRules
metadata:
  name: rule-example
spec:
  rules:
    - macro:
       name: spawned_process
       condition: (evt.type in (execve, execveat) and evt.dir=<)
    - rule:
       name: Linux Cgroup Container Escape Vulnerability (CVE-2022-0492)
       condition: container.id != "" and proc.name = "unshare" and
spawned_process and evt.args contains "mount" and evt.args contains "-o rdma"
and evt.args contains "/release_agent"
       desc: This rule detects an attempt to exploit a container escape
vulnerability in the Linux Kernel.
       output: Detect Linux Cgroup Container Escape Vulnerability (CVE-2022-
0492) (user=%user.loginname uid=%user.loginuid command=%proc.cmdline
args=%proc.args)
       priority: Critical
       tags:
         - process
         - mitre_privilege_escalation
```

Log collection and alerts

DKP exports runtime audit events as Prometheus metrics, allowing you to set up log collection and alerts via resources of the `log-shipper` and `observability` modules. This makes it possible to:

- Connect an external log storage (for example, Loki or Elasticsearch).
- Configure alerts for critical events.

Configuring log and event collection

All runtime audit events are sent to stdout. To collect and forward events to a log storage, create a ClusterLoggingConfig resource following the example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: falco-events
spec:
  destinationRefs:
  - xxxx
  kubernetesPods:
    namespaceSelector:
      labelSelector:
        matchExpressions:
        - key: "kubernetes.io/metadata.name"
          operator: In
          values: [d8-runtime-audit-engine]
  labelFilter:
  - operator: Regex
    values: ["\\{.*"] # Collect only JSON-formatted logs.
    field: "message"
  type: KubernetesPods
```

Configuring critical event alerts

To create alerts for critical events:

1. Make sure the `observability` module is enabled.
2. Create a ClusterObservabilityMetricsRulesGroup or ObservabilityMetricsRulesGroup resource:
   - ClusterObservabilityMetricsRulesGroup: Intended for cluster-wide rules applied to system namespaces.
   - ObservabilityMetricsRulesGroup: Intended for rules that operate within a specific namespace.

Configuration example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterObservabilityMetricsRulesGroup
metadata:
  name: falco-critical-alerts
spec:
  groups:
    - name: falco-critical-alerts
      rules:
        - alert: FalcoCriticalAlertsAreFiring
          for: 1m
          annotations:
            description: |
              There is a suspicious activity on a node {{ $labels.node }}.
              Check your events journal for more details.
            summary: Falco detected a critical security incident.
          expr: |
            sum by (node)
(rate(falcosecurity_falcosidekick_falco_events_total{priority="Critical"}[5m]) > 0)
```

Viewing metrics

To retrieve Prometheus metrics, use the PromQL query `falcosecurity_falcosidekick_falco_events_total{}`:

```
d8 k -n d8-monitoring exec -it prometheus-main-0 prometheus -- \
  curl -s "http://127.0.0.1:9090/api/v1/query?
query=falcosecurity_falcosidekick_falco_events_total" | jq
```

Debugging and simulating events

For debugging and simulating runtime audit events in DKP, you can use:

- The `event-generator` utility.
- The `/test` HTTP endpoint of the `falcosidekick` service.

Enabling debug logging

Falco uses the `debug` logging level by default.

In Falcosidekick, debug logging is disabled by default. To enable it, set the `spec.settings.debugLogging` parameter to `true`, for example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: runtime-audit-engine
spec:
  version: 1
  enabled: true
  settings:
    debugLogging: true
```

Simulating events

Falco

The `event-generator` utility allows you to generate various suspicious actions (for example, system calls or Kubernetes API audit events).

Run the following command to generate a test set of events in a Kubernetes cluster:

```
d8 k run falco-event-generator --image=falcosecurity/event-generator run
```

If you need to simulate a specific action, refer to the utility's documentation.

Falcosidekick

To simulate sending test events to the `falcosidekick` service, use its `/test` HTTP endpoint:

1. Create a test event by running:

   ```
   nsenter -t $(pidof falcosidekick) curl -X POST -H "Content-Type: application/json" -H
   "Accept: application/json" http://localhost:2801/test
   ```

2. Check the event metric:

   ```
   d8 k -n d8-monitoring exec -it prometheus-main-0 prometheus -- \
     curl -s "http://127.0.0.1:9090/api/v1/query?
   query=falcosecurity_falcosidekick_falco_events_total" \
     | jq '.data.result.[] | select (.metric.priority_raw == "debug")'
   ```

   Example output:

```
{
  "metric": {
    "__name__": "falcosecurity_falcosidekick_falco_events_total",
    "container": "kube-rbac-proxy",
    "hostname": "falcosidekick",
    "instance": "192.168.208.7:4212",
    "job": "runtime-audit-engine",
    "node": "dev-master-0",
    "priority": "1",
    "priority_raw": "debug",
    "rule": "Test rule",
    "source": "internal",
    "tier": "cluster"
  },
  "value": [
    1744234729.799,
    "1"
  ]
}
```


Additional resources

- Module `control-plane-manager` documentation
- Module `log-shipper` documentation
- Module `observability` documentation
- Module `runtime-audit-engine` documentation

Security policies

Deckhouse Kubernetes Platform (DKP) lets you manage application security in the cluster using a set of policies that follow the Kubernetes Pod Security Standards model and can be extended with DKP's built-in mechanisms.

DKP implements security policies using Gatekeeper.

Applying Pod Security Standards

DKP supports three security policy levels:

- `privileged`: An unrestricted policy with the broadest possible permissions.
- `baseline`: A minimally restrictive policy that prevents the most well-known and common privilege escalation techniques. Allows the use of a standard (minimally specified) Pod configuration.
- `restricted`: A highly restrictive policy with the strictest requirements for Pods.

Default policy

The default policy is determined as follows:

- In DKP versions prior to v1.55, the default policy is `privileged`.
- Starting from DKP v1.55, the default policy is `baseline`.

> **i** When upgrading DKP to v1.55 or later, the default policy will not change automatically.

Assigning a policy

You can assign a policy in the following ways:

- Globally, using the `settings.podSecurityStandards.defaultPolicy` parameter of the `admission-policy-engine` module.
- Per namespace, using the `security.deckhouse.io/pod-policy=<POLICY_NAME>` label.

  Example command to assign the `restricted` policy to all Pods in the `my-namespace` namespace:

```
d8 k label ns my-namespace security.deckhouse.io/pod-policy=restricted
```

Enforcement modes

Allowed policy enforcement modes:

- `deny`: Blocks actions from being executed.
- `dryrun`: Does not affect execution and used for debugging. Event information can be viewed in Grafana or in the console using `kubectl`.
- `warn`: Works like `dryrun` but also displays a warning with the reason the action would have been denied in `deny` mode.

By default, Pod Security Standards policies in DKP are enforced in `deny` mode. In this mode, application Pods that do not comply with the policies cannot be run in the cluster.

As with policy assignment, enforcement mode can be set:

- Globally, using the `settings.podSecurityStandards.enforcementAction` parameter of the `admission-policy-engine` module.
- Per namespace, using the `security.deckhouse.io/pod-policy-action=<POLICY_ACTION>` label.

  Example command to set the `warn` mode for all Pods in the `my-namespace` namespace:

  ```
  d8 k label ns my-namespace security.deckhouse.io/pod-policy-action=warn
  ```

Extending a policy

You can extend the `baseline` and `restricted` policies using Gatekeeper templates by adding extra checks to the existing ones.

To extend a policy:

1. Create a validation template using a `ConstraintTemplate`.
2. Apply the template to the `baseline` or `restricted` policy.

Example template for validating the container image repository address:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sallowedrepos
spec:
  crd:
    spec:
      names:
        kind: K8sAllowedRepos
      validation:
        openAPIV3Schema:
          type: object
          properties:
            repos:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.pod_security_standards.extended

        violation[{"msg": msg}] {
          container := input.review.object.spec.containers[_]
          satisfied := [good | repo = input.parameters.repos[_] ; good =
startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>, allowed repos
are %v", [container.name, container.image, input.parameters.repos])
        }

        violation[{"msg": msg}] {
          container := input.review.object.spec.initContainers[_]
          satisfied := [good | repo = input.parameters.repos[_] ; good =
startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>, allowed repos
are %v", [container.name, container.image, input.parameters.repos])
        }
```

Example of applying the template to the `restricted` policy:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: prod-repo
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaceSelector:
      matchLabels:
        security.deckhouse.io/pod-policy: restricted
  parameters:
    repos:
      - "mycompany.registry.com"
```

In this example, the repository address in the `image` field of all Pods in namespaces labeled `security.deckhouse.io/pod-policy: restricted` is checked. If the `image` address of a created Pod does not start with `mycompany.registry.com`, the Pod will not be created.

Helpful resources for creating extended policies:

- Custom policy examples.
- Gatekeeper documentation contains information on templates and policy language.
- Gatekeeper library contains examples of validation templates.

Operational policies

DKP provides a mechanism for creating operational policies using the OperationPolicy. Operational policies define requirements for cluster objects such as allowed repositories, required resources, probes, and more.

The DKP development team recommends applying the following minimal operational policy:

```
apiVersion: deckhouse.io/v1alpha1
kind: OperationPolicy
metadata:
  name: common
spec:
  policies:
    allowedRepos:
      - myrepo.example.com
      - registry.deckhouse.io
    requiredResources:
      limits:
        - memory
      requests:
        - cpu
        - memory
    disallowedImageTags:
      - latest
    requiredProbes:
      - livenessProbe
      - readinessProbe
    maxRevisionHistoryLimit: 3
    imagePullPolicy: Always
    priorityClassNames:
    - production-high
    - production-low
    checkHostNetworkDNSPolicy: true
    checkContainerDuplicates: true
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          operation-policy.deckhouse.io/enabled: "true"
```

This policy defines basic operational requirements for cluster objects, including allowed container registries, required resources and probes, restrictions on using images with the `latest` tag, allowed priority classes, and other settings that improve application security and stability.

To assign this operational policy, add the `operation-policy.deckhouse.io/enabled=true` label to the target namespace:

```
d8 k label ns my-namespace operation-policy.deckhouse.io/enabled=true
```

Security policies

Using the SecurityPolicy, you can create security policies that define container behavior restrictions in the cluster, such as host network access, privileges, AppArmor usage, and more.

> **i** Detailed information on pod and container security settings (such as `hostNetwork`, `hostPID`, `hostIPC` and others), available values, and practical suggestions are available on the “Pod and container security settings” page.

Example security policy:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: mypolicy
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: true
    allowHostNetwork: true
    allowHostPID: false
    allowPrivileged: false
    allowPrivilegeEscalation: false
    allowedFlexVolumes:
    - driver: vmware
    allowedHostPorts:
    - max: 4000
      min: 2000
    allowedProcMount: Unmasked
    allowedAppArmor:
    - unconfined
    allowedUnsafeSysctls:
    - kernel.*
    allowedVolumes:
    - hostPath
    - projected
    fsGroup:
      ranges:
      - max: 200
        min: 100
      rule: MustRunAs
    readOnlyRootFilesystem: true
    requiredDropCapabilities:
    - ALL
    runAsGroup:
      ranges:
      - max: 500
        min: 300
      rule: RunAsAny
    runAsUser:
      ranges:
      - max: 200
        min: 100
      rule: MustRunAs
    seccompProfiles:
      allowedLocalhostFiles:
      - my_profile.json
      allowedProfiles:
      - Localhost
    supplementalGroups:
      ranges:
      - max: 133
        min: 129
      rule: MustRunAs
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          enforce: mypolicy
```

To assign this security policy, add the `enforce: "mypolicy"` label to the target namespace.

Partial policy enforcement

To enforce specific security policies without disabling the entire predefined set, follow these steps:

1. Add the `security.deckhouse.io/pod-policy: privileged` label to the target namespace to disable the built-in policy set.
2. Create a SecurityPolicy that matches the `baseline` or `restricted` level. In the `policies` section, specify only the security settings you need.
3. Add an extra label to the namespace matching the `namespaceSelector` in the SecurityPolicy.

Example SecurityPolicy configuration for the `baseline` level:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: baseline
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: false
    allowHostNetwork: false
    allowHostPID: false
    allowPrivilegeEscalation: true
    allowPrivileged: false
    allowedAppArmor:
      - runtime/default
      - localhost/*
    allowedCapabilities:
      - AUDIT_WRITE
      - CHOWN
      - DAC_OVERRIDE
      - FOWNER
      - FSETID
      - KILL
      - MKNOD
      - NET_BIND_SERVICE
      - SETFCAP
      - SETGID
      - SETPCAP
      - SETUID
      - SYS_CHROOT
    allowedHostPaths: []
    allowedHostPorts:
      - max: 0
        min: 0
    allowedProcMount: Default
    allowedUnsafeSysctls:
      - kernel.shm_rmid_forced
      - net.ipv4.ip_local_port_range
      - net.ipv4.ip_unprivileged_port_start
      - net.ipv4.tcp_syncookies
      - net.ipv4.ping_group_range
      - net.ipv4.ip_local_reserved_ports
      - net.ipv4.tcp_keepalive_time
      - net.ipv4.tcp_fin_timeout
      - net.ipv4.tcp_keepalive_intvl
      - net.ipv4.tcp_keepalive_probes
    seLinux:
      - type: ""
      - type: container_t
      - type: container_init_t
      - type: container_kvm_t
      - type: container_engine_t
```

```
    seccompProfiles:
      allowedProfiles:
        - RuntimeDefault
        - Localhost
        - undefined
        - ''
      allowedLocalhostFiles:
        - '*'
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          operation-policy.deckhouse.io/baseline-enabled: "true"
```

Example SecurityPolicy configuration for the `restricted` level:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: restricted
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: false
    allowHostNetwork: false
    allowHostPID: false
    allowPrivilegeEscalation: false
    allowPrivileged: false
    allowedAppArmor:
      - runtime/default
      - localhost/*
    allowedCapabilities:
      - NET_BIND_SERVICE
    allowedHostPaths: []
    allowedHostPorts:
      - max: 0
        min: 0
    allowedProcMount: Default
    allowedUnsafeSysctls:
      - kernel.shm_rmid_forced
      - net.ipv4.ip_local_port_range
      - net.ipv4.ip_unprivileged_port_start
      - net.ipv4.tcp_syncookies
      - net.ipv4.ping_group_range
      - net.ipv4.ip_local_reserved_ports
      - net.ipv4.tcp_keepalive_time
      - net.ipv4.tcp_fin_timeout
      - net.ipv4.tcp_keepalive_intvl
      - net.ipv4.tcp_keepalive_probes
    allowedVolumes:
      - configMap
      - csi
      - downwardAPI
      - emptyDir
      - ephemeral
      - persistentVolumeClaim
      - projected
      - secret
    requiredDropCapabilities:
      - ALL
    runAsUser:
      rule: MustRunAsNonRoot
```

```
    seLinux:
      - type: ""
      - type: container_t
      - type: container_init_t
      - type: container_kvm_t
      - type: container_engine_t
    seccompProfiles:
      allowedProfiles:
        - RuntimeDefault
        - Localhost
      allowedLocalhostFiles:
        - '*'
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          operation-policy.deckhouse.io/restricted-enabled: "true"
```

Gatekeeper custom resources

Gatekeeper offers advanced capabilities for modifying Kubernetes resources using mutation policies. These policies are defined through the following custom resources:

- AssignMetadata: Modifies the `metadata` section of a resource.
- Assign: Modifies fields other than `metadata`.
- ModifySet: Adds or removes values from a list, such as container run arguments.
- AssignImage: Modifies the `image` field of a resource.

For more on modifying Kubernetes resources using mutation policies, refer to the Gatekeeper documentation.

Custom Gatekeeper policy examples

Here you can find examples of Gatekeeper policies that let you extend the standard cluster security mechanism.

Preventing deletion of a node without a specified label

> **i** The `DELETE` operations are handled by Gatekeeper by default.

You can create a Gatekeeper policy to prevent a node deletion unless it has a special label assigned.

In the following example, the `oldObject` field is used to check labels on the node being deleted:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: d8customnodedeleteguard
spec:
  crd:
    spec:
      names:
        kind: D8CustomNodeDeleteGuard
      validation:
        openAPIV3Schema:
          type: object
          properties:
            requiredLabelKey:
              type: string
            requiredLabelValue:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.custom

        is_delete { input.review.operation == "DELETE" }
        is_node { input.review.kind.kind == "Node" }

        has_required_label {
          key := input.parameters.requiredLabelKey
          val := input.parameters.requiredLabelValue
          obj := input.review.oldObject
          obj.metadata.labels[key] == val
        }

        violation[{"msg": msg}] {
          is_delete
          is_node
          not has_required_label
          msg := sprintf("Node deletion is blocked. Add label %q=%q to proceed.",
[input.parameters.requiredLabelKey, input.parameters.requiredLabelValue])
        }
---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: D8CustomNodeDeleteGuard
metadata:
  name: require-node-delete-label
spec:
  enforcementAction: warn
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Node"]
  parameters:
    requiredLabelKey: "admission.deckhouse.io/allow-delete"
    requiredLabelValue: "true"
```

Preventing exec and attach operations to specific Pods

The `admission-policy-engine` module's webhook routes `CONNECT` requests for `pods/exec` and `pods/attach` through Gatekeeper. This allows custom policies to allow or deny `kubectl exec` and `kubectl attach` operations.

Built-in policy for `heritage: deckhouse` label on Pods

To protect system components managed by Deckhouse, the `admission-policy-engine` module includes a built-in policy `D8DenyExecHeritage` that forbids running `kubectl exec` and `kubectl attach` operations to all Pods with the `heritage: deckhouse` label.

This policy doesn't apply to the following users who are allowed to run `kubectl exec` and `kubectl attach` operations to Pods labeled with `heritage: deckhouse`:

- `system:sudouser`
- service accounts from `d8-*` namespaces (`system:serviceaccount:d8-*`)
- service accounts from `kube-*` namespaces (`system:serviceaccount:kube-*`)

Custom policy example

You can create your own Gatekeeper policy to deny `kubectl exec` and `kubectl attach` operations in specific namespaces. In the following example, `input.review.operation` and `input.review.resource.resource` are used to check for `CONNECT` operations:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: d8customdenyexec
spec:
  crd:
    spec:
      names:
        kind: D8CustomDenyExec
      validation:
        openAPIV3Schema:
          type: object
          properties:
            forbiddenNamespaces:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.custom

        is_connect {
          input.review.operation == "CONNECT"
        }

        # requestSubResource is preferred, but fall back to subResource for older APIs
        subresource_is(sub) {
          sr := object.get(input.review, "requestSubResource",
input.review.subResource)
          sr == sub
        }

        is_exec_or_attach {
          input.review.resource.resource == "pods"
          subresource_is("exec")
        }

        is_exec_or_attach {
          input.review.resource.resource == "pods"
          subresource_is("attach")
        }

        is_forbidden_namespace {
          ns := input.review.namespace
          ns == input.parameters.forbiddenNamespaces[_]
        }
```

```
      violation[{"msg": msg}] {
        is_connect
        is_exec_or_attach
        is_forbidden_namespace
        msg := sprintf("Exec/attach is forbidden in namespace %q",
[input.review.namespace])
      }
---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: D8CustomDenyExec
metadata:
  name: deny-exec-in-namespaces
spec:
  enforcementAction: deny
  match:
    kinds:
      - apiGroups: ["*"]
        kinds: ["*"]
    scope: Namespaced
  parameters:
    forbiddenNamespaces:
      - production
      - staging
```

Key data and checks for `CONNECT` validation:

- Use `input.review.operation == "CONNECT"` to check for `CONNECT` operations.
- User information is available in `input.review.userInfo.username` and `input.review.userInfo.groups`.
- The namespace is available in `input.review.namespace`.

Image signature verification

DKP supports container image signature verification using Cosign.
Container image signature verification allows you to ensure their integrity (that the image has not been modified since its creation) and authenticity (that the image was created by a trusted source). You can enable container image signature verification in the cluster using the policies.verifyImageSignatures parameter of the SecurityPolicy.

Images are signed by creating a special tag in the container registry that contains the image signature. The signature is generated for the digest (hash) of your image.
If your image is `my-repo/app:latest` with the hash `sha256:abc123EXAMPLE`, the tag `my-repo/app:sha256-abc123EXAMPLE.sig` will appear in the image store.

Therefore, the image signing process consists of calculating and publishing an additional tag to the container registry, without modifying the existing image.
After signing the image, there is no need to push it to the image store again. You only need to log in to the container registry with write access.

To sign an image with Cosign, do the following:

1. Make sure that Cosign version 2 or lower is used:

   Check the version: `cosign version`.

   ```
   cosign version
   ```

2. Generate a key pair (public and private):

   ```
   cosign generate-key-pair
   ```

3. Sign the image in the container registry using the generated private key:

   ```
   cosign sign --key <KEY> <REGISTRY_IMAGE_PATH>
   ```

   Here:

- `<REGISTRY_IMAGE_PATH>` is the path to the image that needs to be specified at startup, for example: registry.private.com/labs/application/image:latest.

To enable container image signature verification in a DKP cluster:

1. Use the `policies.verifyImageSignatures` parameter in SecurityPolicy and specify the generated public key.

   Example SecurityPolicy configuration for verifying signatures of container images in `registry.private.ru`, located under `/labs/application/`:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: SecurityPolicy
   metadata:
     name: verify-image-test
   spec:
     enforcementAction: Deny
     match:
       namespaceSelector:
         labelSelector:
           matchLabels:
             example-security-policy/enabled: true
     policies:
       allowHostIPC: true
       allowHostNetwork: true
       allowHostPID: false
       allowPrivilegeEscalation: true
       allowPrivileged: false
       allowRbacWildcards: true
       verifyImageSignatures:
         - publicKeys:
             - |-
               -----BEGIN PUBLIC KEY-----
               ...
               -----END PUBLIC KEY-----
           reference: registry.private.ru/labs/application/*
   ```

   The label name specified in `match.namespaceSelector.labelSelector.matchLabels` can be any name. It only needs to match between the policy selector and the corresponding namespace.

   More details about selector usage are available in the selector setup description.

2. Create an OperationPolicy that restricts running pods from third-party registries:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: OperationPolicy
   metadata:
     name: test-operation-policy
   spec:
     enforcementAction: Deny
     match:
       namespaceSelector:
         labelSelector:
           matchLabels:
             example-operation-policy/enabled: "true"
     policies:
       allowedRepos:
       - registry.private.ru
   ```

   The label name specified in `match.namespaceSelector.labelSelector.matchLabels` can be any name. It only needs to match between the policy selector and the corresponding namespace.

   More details about selector usage are available in the selector setup description.

3. Add a label to the namespace where signature verification should be enabled (specify your namespace):

```
d8 k label ns <NAMESPACE> example-security-policy/enabled=true
```

4. Add a label to the namespace where running pods from third-party registries should be restricted (specify your namespace):

   ```
   d8 k label ns <NAMESPACE> example-operation-policy/enabled=true
   ```

5. To verify how image signing works, deploy pods in the namespace with signed and unsigned images (specify your namespace):

   ```
   d8 k -n <NAMESPACE> run signed-pod --image=<SIGNED_IMAGE>
   d8 k -n <NAMESPACE> run unsigned-pod --image=<UNSIGNED_IMAGE>
   ```

According to this policy, if any container image address matches the `reference` parameter value and the image is unsigned, or the signature does not match the specified keys, pod creation will be denied.

Example error output when creating a pod with an image that fails signature verification:

```
[verify-image-signatures] Image signature verification failed: nginx:1.17.2
```

Using alternative security policy management tools

If you use an alternative solution for security policy management in a DKP cluster (for example, Kyverno), configure exceptions for the following namespaces:

- `kube-system`
- all namespaces with the `d8-*` prefix (for example, `d8-system`)

Without these exceptions, policies may block or disrupt system component operations.

Additional resources

- Module `admission-policy-engine` documentation

Container image vulnerability scanning

Deckhouse Kubernetes Platform (DKP) provides a built-in tool for automatically detecting vulnerabilities in container images based on the Trivy project.

Vulnerability scanning

DKP scans all container images used by cluster pods. The checks include:

- Known vulnerabilities (CVEs) in the images in use.
- Compliance with CIS benchmarks (compliance checks).

Scanning uses public vulnerability databases.

Scanning in namespaces

By default, only the `default` namespace is scanned.

To enable scanning for a specific namespace, add the label `security-scanning.deckhouse.io/enabled=""` to it.

As soon as at least one namespace with this label is detected in the cluster, scanning of the `default` namespace stops. To re-enable scanning for the `default` namespace, add the label with the following command:

```
d8 k label namespace default security-scanning.deckhouse.io/enabled=""
```

In the current version, there is no option to limit the list of resources to be scanned within a namespace. DKP scans **all resources** in a namespace labeled with `security-scanning.deckhouse.io/enabled=""`.

Start conditions and scanning process

Scanning starts:

- automatically every 24 hours,
- when components with new images are deployed in namespaces with scanning enabled.

Scanning occurs according to the following process:

1. In the namespace of each scanned resource, a VulnerabilityReport object is created.
2. This object contains the annotation `trivy-operator.aquasecurity.github.io/report-ttl`, which specifies the report's time-to-live (default: `24h`).
3. When the TTL expires, the VulnerabilityReport object is deleted, and the scan is run again.

Forcing a rescan

To manually trigger a rescan of a resource, you can use either of the following methods:

- Update the `trivy-operator.aquasecurity.github.io/report-ttl` annotation with a short time-to-live value.

  Example:

  ```
  d8 k annotate VulnerabilityReport -n <NAMESPACE> <REPORT_NAME> trivy-
  operator.aquasecurity.github.io/report-ttl=1s --overwrite
  ```

- Delete the VulnerabilityReport object from the namespace containing the scanned resource.

  Example:

  ```
  d8 k delete VulnerabilityReport -n <NAMESPACE> <REPORT_NAME>
  ```

For details on how to view scan results, see Scanning container images for vulnerabilities.

Certificate management

Deckhouse Kubernetes Platform (DKP) provides built-in tools for managing TLS certificates in the cluster and supports:

- Requesting certificates from all supported sources, such as Let's Encrypt, HashiCorp Vault, Venafi.
- Issuing self-signed certificates.
- Automatic renewal and expiration monitoring.
- Deploying `cm-acme-http-solver` on master nodes and dedicated nodes.

This page describes the available certificate management features in DKP and the procedure for working with certificate issuers.

> **i** Examples of certificate configuration, usage of the `tls-acme` annotation, and secure handling of credentials are described in Using TLS certificates.

Monitoring

DKP exports metrics to Prometheus, allowing you to monitor:

- Certificate expiration dates
- Certificate renewal status

Access roles

DKP provides several predefined roles for accessing resources:

Role	Permissions
`User`	View Certificate and Issuer resources in available namespaces, as well as global ClusterIssuer resources.
`Editor`	Manage Certificate and Issuer resources in available namespaces.

Role	Permissions
`ClusterEditor`	Manage Certificate and Issuer resources in all namespaces.
`SuperAdmin`	Manage internal system objects.

Working with certificate issuers

DKP supports the following default certificate issuers (ClusterIssuer):

- `letsencrypt`: Issues TLS certificates using the public CA Let’s Encrypt and ACME HTTP validation. Suitable for automatically obtaining trusted certificates for most public services. Configuration details can be found in the official `cert-manager` documentation.
- `letsencrypt-staging`: Similar to `letsencrypt`, but uses Let’s Encrypt’s staging server. Useful for debugging configuration and testing the certificate issuance process. More details available in the Let’s Encrypt staging docs.
- `selfsigned`: Issues self-signed certificates. Useful when external trust is not required (for example, for internal services).
- `selfsigned-no-trust`: Also issues self-signed certificates but without automatically adding the root certificate to the trust store. Suitable for manual trust management.

In some cases, additional types of ClusterIssuer may be needed:

- If you want to use a Let’s Encrypt certificate with DNS validation via a third-party DNS provider.
- if you need to use a certificate authority (CA) other than Let’s Encrypt. See the full list of supported CAs in the `cert-manager` documentation.

Adding a ClusterIssuer with DNS-01 validation via webhook

To verify domain ownership with Let’s Encrypt using the `DNS-01` method, the `cert-manager` module must be able to create TXT records in the DNS zone associated with the domain.

`cert-manager` includes built-in support for popular DNS providers such as AWS Route53, Google Cloud DNS, Cloudflare, and others. A full list is available in the official `cert-manager` documentation.

If your provider is not directly supported, you can configure a webhook and deploy a custom ACME handler in the cluster that performs the necessary DNS record updates.

The following example is based on using Yandex Cloud DNS:

1. To handle the webhook, deploy the `Yandex Cloud DNS ACME webhook` service in the cluster according to the official documentation.
2. Create a ClusterIssuer resource using the following example:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: yc-clusterissuer
  namespace: default
spec:
  acme:
    # Replace this email address with your own.
    # Let's Encrypt will use it to notify you about expiring certificates
    # and issues related to your account.
    email: your@email.com
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      # The Secret resource used to store the account private key.
      name: secret-ref
    solvers:
      - dns01:
          webhook:
            config:
              # The folder ID containing your DNS zone.
              folder: <your-folder-ID>
              # Secret used to access the service account.
              serviceAccountSecretRef:
                name: cert-manager-secret
                key: iamkey.json
            groupName: acme.cloud.yandex.com
            solverName: yandex-cloud-dns
```

Adding a ClusterIssuer that uses a custom certificate authority (CA)

1. Generate the certificate:

   ```
   openssl ecparam -name prime256v1 -genkey -out rootCAKey.pem
   openssl req -x509 -sha256 -new -nodes -key rootCAKey.pem -days 3650 -out rootCACert.pem
   ```

2. Create a secret in the `d8-cert-manager` namespace with an arbitrary name, containing the certificate files.

   - Example of creating a secret using the `d8 k` command:

     ```
     d8 k create secret tls internal-ca-key-pair -n d8-cert-manager --key="rootCAKey.pem" --
     cert="rootCACert.pem"
     ```

   - Example of creating a secret from a YAML file (the certificate file contents must be Base64-encoded):

     ```
     apiVersion: v1
     data:
       tls.crt: <output of `cat rootCACert.pem | base64 -w0`>
       tls.key: <output of `cat rootCAKey.pem | base64 -w0`>
     kind: Secret
     metadata:
       name: internal-ca-key-pair
       namespace: d8-cert-manager
     type: Opaque
     ```

3. Create a ClusterIssuer with any name using the created secret:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: inter-ca
spec:
  ca:
    secretName: internal-ca-key-pair   # Name of the created secret.
```

You can now use the created ClusterIssuer to issue certificates for all DKP components or for a specific component.

For example, to use this ClusterIssuer for issuing certificates for all DKP components, set its name in the global parameter `clusterIssuerName`:

```
spec:
  settings:
    modules:
      https:
        certManager:
          clusterIssuerName: inter-ca
        mode: CertManager
      publicDomainTemplate: '%s.<public_domain_template>'
  version: 1
```

Adding an Issuer and ClusterIssuer using HashiCorp Vault to request certificates

To configure certificate issuance using Vault, refer to the HashiCorp documentation.

After configuring the PKI and enabling Kubernetes authentication, follow these steps:

1. Create a ServiceAccount and copy the reference to its secret:

   ```
   d8 k create serviceaccount issuer

   ISSUER_SECRET_REF=$(d8 k get serviceaccount issuer -o json | jq -r ".secrets[].name")
   ```

2. Create the Issuer resource:

   ```
   d8 k apply -f - <<EOF
   apiVersion: cert-manager.io/v1
   kind: Issuer
   metadata:
     name: vault-issuer
     namespace: default
   spec:
     vault:
       server: http://vault.default.svc.cluster.local:8200
       # Defined during PKI configuration.
       path: pki/sign/example-dot-com
       auth:
         kubernetes:
           mountPath: /v1/auth/kubernetes
           role: issuer
           secretRef:
             name: $ISSUER_SECRET_REF
             key: token
   EOF
   ```

3. Create the Certificate resource to obtain a TLS certificate signed by Vault CA:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com
  namespace: default
spec:
  secretName: example-com-tls
  issuerRef:
    name: vault-issuer
  # Domains must match those configured in the PKI in Vault.
  commonName: www.example.com
  dnsNames:
  - www.example.com
EOF
```

Additional resources

- Global parameters
- Module `cert-manager` documentation

Integration with software and security solutions

Deckhouse Kubernetes Platform (DKP) supports integration with Kaspersky Unified Monitoring and Analysis Platform (KUMA), a unified monitoring and analysis system by Kaspersky Lab. As part of the integration, security events and audit logs from the cluster are sent to KUMA for further analysis.

Sending logs to KUMA

To send logs to KUMA, configure log collection and delivery in DKP using the following resources:

- ClusterLogDestination: Defines log storage parameters.
- ClusterLoggingConfig: Defines cluster log collection parameters.

> **i** On the KUMA side, configure the appropriate resources for receiving events.

Example ClusterLogDestination and ClusterLoggingConfig configurations

- Sending logs in JSON format over UDP:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-udp-json
spec:
  type: Socket
  socket:
    address: IP_ADDRESS:PORT # Replace as needed.
    mode: UDP
    encoding:
      codec: "JSON"
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
    - /var/log/kube-audit/audit.log
  destinationRefs:
    - kuma-udp-json
```

- Sending logs in JSON format over TCP:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-tcp-json
spec:
  type: Socket
  socket:
    address: IP_ADDRESS:PORT # Replace as needed.
    mode: TCP
    tcp:
      verifyCertificate: false
      verifyHostname: false
    encoding:
      codec: "JSON"
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
    - /var/log/kube-audit/audit.log
  destinationRefs:
    - kuma-tcp-json
```

- Sending logs in CEF format over TCP:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-tcp-cef
spec:
  type: Socket
  socket:
    extraLabels:
      cef.name: d8
      cef.severity: "1"
    address: IP_ADDRESS:PORT # Replace as needed.
    mode: TCP
    tcp:
      verifyCertificate: false
      verifyHostname: false
    encoding:
      codec: "CEF"
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
    - /var/log/kube-audit/audit.log
  logFilter:
    - field: userAgent
      operator: Regex
      values: [ "kubelet.*" ]
  destinationRefs:
    - kuma-tcp-cef
```

- Sending logs in Syslog format over TCP:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-tcp-syslog
spec:
  type: Socket
  socket:
    address: IP_ADDRESS:PORT # Replace as needed.
    mode: TCP
    tcp:
      verifyCertificate: false
      verifyHostname: false
    encoding:
      codec: "Syslog"
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
    - /var/log/kube-audit/audit.log
  logFilter:
    - field: userAgent
      operator: Regex
      values: [ "kubelet.*" ]
  destinationRefs:
    - kuma-tcp-syslog
```

- Sending logs to Apache Kafka:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: kuma-kafka
spec:
  type: Kafka
  kafka:
    bootstrapServers:
      - kafka-address:9092 # Replace as needed.
    topic: k8s-logs
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  destinationRefs:
    - kuma-kafka
  file:
    include:
      - /var/log/kube-audit/audit.log
  logFilter:
    - field: userAgent
      operator: Regex
      values:
        - kubelet.*
  type: File
```

KCS configuration features

Kaspersky Container Security (KCS) requires extended access rights to the OS kernel in order to monitor the execution environment. Specifically, the `kcs-ih` (Image Hub) component must run in privileged mode. To do this, certain process privilege management mechanisms at the Linux OS kernel level must be disabled. For more details, see the Kaspersky Container Security documentation.

To enable KCS runtime monitoring in a DKP cluster, assign the label `security.deckhouse.io/pod-policy=privileged` to the namespace where KCS components are located.

Example command for assigning a label:

```
d8 k label namespace kcs security.deckhouse.io/pod-policy=privileged
```

Antivirus scanning exclusions for nodes

If antivirus software is installed on DKP cluster nodes (for example, Kaspersky Endpoint Security for Linux, KESL), you may need to exclude Deckhouse service directories from scanning to avoid false positives.

List of Deckhouse service directories (also available CSV format):

Directory	Purpose
`/etc/cni/` (any node)	CNI module configuration files
`/etc/kubernetes` (any node)	Static pod manifests, PKI certificate files
`/mnt/kubernetes-data` (master node)	Only present in clusters deployed in the cloud when a separate disk is used for the etcd database
`/mnt/vector-data` (any node, `log-shipper` module)	Service data for log delivery status
`/opt/cni/bin/` (any node)	CNI module executables
`/opt/deckhouse/bin/` (any node)	Executables required for Deckhouse operation
`/var/lib/bashible` (any node)	Node configuration files
`/var/lib/containerd` (any node)	Stores CRI-related data (for example, containerd). Contains container image layers, filesystem snapshots, metadata, logs, and other container information
`/var/lib/deckhouse/` (master node)	Deckhouse module files dynamically loaded from the image registry
`/var/lib/etcd` (master node)	etcd database
`/var/lib/kubelet/` (any node)	kubelet configuration files
`/var/lib/upmeter` (master node, `upmeter` module)	`upmeter` module database
`/var/log/containers` (any node)	Container logs (when using containerd)
`/var/log/pods/` (any node)	Logs of all pod containers running on the node

KESL configuration recommendations

To ensure DKP functions correctly with KESL installed, follow these steps:

1. Disable the following KESL tasks:
   - `Firewall Management (ID: 12)`
   - `Web Threat Protection (ID: 14)`
   - `Network Threat Protection (ID: 17)`
   - `Web Control (ID: 26)`

> **i** The list of tasks may differ in future KESL versions.

2. Make sure node resources meet the requirements of:
    - DKP
    - KESL
3. For performance optimization, follow the official Kaspersky recommendations.

Additional resources

- Module `log-shipper` custom resources
- Module `upmeter` documentation

Automatic distribution of secrets across namespaces

Deckhouse Kubernetes Platform (DKP) lets you automatically distribute secrets across all namespaces in the cluster. This helps eliminate the need to manually copy secrets in CI/CD pipelines — for example, when pulling container images or working with Ceph RBD storage.

How it works

- Deckhouse monitors secrets in the `default` namespace that have the special label `secret-copier.deckhouse.io/enabled: ""`.
- These secrets are automatically copied to all namespaces in the cluster.
- When the original secret in the `default` namespace is updated, the new content is propagated to all namespaces.
- If the original secret in `default` is deleted, all of its copies in other namespaces are also deleted.
- If a secret copy is manually modified in a namespace, it will be automatically overwritten with the content of the original secret from `default`.
- When new namespaces are created, all secrets from `default` labeled with `secret-copier.deckhouse.io/enabled: ""` are automatically copied into them.
- There’s a regular nightly synchronization that ensures all secrets remain up to date.

Incompatibility with multitenancy mode

The automatic secret distribution mechanism is not compatible with the multitenancy mode.

Multitenancy mode creates isolated environments for users within projects. Automatically distributing secrets to all namespaces may violate this security model. If sensitive data accidentally reaches a user’s isolated environment, it may lead to data exposure.

If you need to provide a shared certificate (for example, a wildcard certificate for an internal environment) or a shared registry access token, do not use automatic distribution. Instead, include such secrets in the project template. A cluster administrator must define them in the project configuration.

Configuring automatic secret distribution

1. Create a Secret resource in the `default` namespace.
2. Add the label `secret-copier.deckhouse.io/enabled: ""`.
3. Apply the resource using the `d8 k apply` command. It will then be automatically distributed to all namespaces.

> ⚠ Only secrets from the `default` namespace are automatically distributed. Secrets with the label `secret-copier.deckhouse.io/enabled: ""` created in any other namespace will be automatically deleted.

Distributing secrets to selected namespaces

To copy a secret to specific namespaces only, use the annotation with a label selector `secret-copier.deckhouse.io/target-namespace-selector: "app=custom"`. In this case, the secret will be copied only to namespaces that match the specified selector.

Monitoring in Deckhouse Kubernetes Platform

Deckhouse Kubernetes Platform (DKP) provides a Kubernetes monitoring solution based on **Prometheus** and **Grafana**. DKP automatically configures metrics collection in the cluster from nodes, pods, and key cluster components (etcd, kube-apiserver, CoreDNS), which enables preset dashboards for analyzing CPU, memory, disk, and network usage.

Cluster monitoring is enabled by default in the `Default` and `Managed` module bundles.

All components, including Prometheus and Alertmanager, operate in a fault-tolerant mode and can be used in cloud environments and on bare-metal servers.

The principles of Prometheus operation is covered in Configuring a system for collecting and storing metrics.

Several types of monitoring are implemented in DKP:

- Hardware resource monitoring
- Kubernetes monitoring
- Ingress monitoring
- Control plane monitoring
- Network interaction monitoring
- Extended monitoring
- Cluster SLA monitoring

DKP includes an alerting system that supports sending event notifications, including to external systems.

Hardware resource monitoring

Tracking of cluster hardware resource capacity is provided with graphs showing utilization of:

- CPU
- Memory
- Disk
- Network

Graphs are available with aggregation by:

- Pods
- Controllers
- Namespaces
- Nodes

Kubernetes monitoring

The module `monitoring-kubernetes` is designed for basic cluster node monitoring.

It provides secure metrics collection and offers a basic set of rules for monitoring:

- Current container runtime version (docker, containerd) on the node and its compliance with versions allowed for use.
- Overall cluster monitoring subsystem health (Dead man’s switch).
- Available file descriptors, sockets, free space, and inodes.
- Operation of `kube-state-metrics`, `node-exporter`, `kube-dns`.
- Cluster node state (NotReady, drain, cordon).
- Time synchronization state on nodes.
- Cases of prolonged CPU steal exceeding.
- Conntrack table state on nodes.
- Pods with incorrect state (as a possible consequence of kubelet issues) and more.

Ingress monitoring

Statistics collection for `ingress-nginx` in Prometheus is implemented with detailed metrics (response time, codes, geography, etc.), available in different dimensions (namespace, vhost, ingress). Data is visualized in Grafana with interactive dashboards. Detailed description is available in the section about Ingress monitoring.

The module is enabled by default in the `Default` and `Managed` module bundles.

Disabling collection of detailed statistics from Ingress resources

By default, DKP collects detailed statistics from all Ingress resources in the cluster, which generates a high load on the monitoring system.

To disable statistics collection, add the label `ingress.deckhouse.io/discard-metrics: "true"` to the corresponding namespace or Ingress resource.

- Example of disabling statistics (metrics) collection for all Ingress resources in the `review-1` namespace:

  ```
  d8 k label ns review-1 ingress.deckhouse.io/discard-metrics=true
  ```

- Example of disabling statistics (metrics) collection for all `test-site` Ingress resources in the `development` namespace:

  ```
  d8 k label ingress test-site -n development ingress.deckhouse.io/discard-metrics=true
  ```

Control plane monitoring

Control plane monitoring is performed using the `monitoring-kubernetes-control-plane` module, which organizes secure metrics collection and provides a basic set of monitoring rules for the following cluster components:

- kube-apiserver
- kube-controller-manager
- kube-scheduler
- kube-etcd

Cluster monitoring

DKP securely collects monitoring metrics and configures rules.

DKP monitoring capabilities:

- Monitoring current container runtime version (containerd) on the node and its compliance with versions allowed for use in DKP.
- Monitoring cluster monitoring subsystem health (“Dead man’s switch”).
- Monitoring available file descriptors, sockets, free space, and inodes.
- Monitoring cluster node state (NotReady, drain, cordon).
- Operation of `kube-state-metrics`, `node-exporter`, `kube-dns`.
- Monitoring time synchronization state on nodes.
- Monitoring cases of prolonged CPU steal exceeding.
- Monitoring Conntrack table state on nodes.
- Monitoring pods with incorrect state (as a possible consequence of kubelet issues).
- Monitoring control plane components (implemented by the `monitoring-kubernetes-control-plane` module).

Extended monitoring mode

DKP supports an extended monitoring mode via the `extended-monitoring` module, allowing you to configure:

- Monitoring secrets in the cluster (Secret objects) and TLS certificate expiration in them.
- Collecting Kubernetes cluster events as metrics.
- Monitoring container image availability in registry used by controllers (Deployments, StatefulSets, DaemonSets, CronJobs).
- Monitoring objects in namespaces that have the `extended-monitoring.deckhouse.io/enabled=""` label.

The module can send alerts based on the following metrics:

- Free space and inodes on node disks
- Node utilization
- Pod and container image availability
- Certificate expiration
- Other cluster events

Alerts

Monitoring in DKP includes event notifications. The standard delivery includes a set of basic warnings covering cluster state and its components. There is also the ability to add custom alerts.

For the list of all available alerts in the DKP monitoring system, refer to the corresponding documentation page.

Sending alerts to external systems

DKP supports sending alerts using Alertmanager:

- Via SMTP protocol
- To PagerDuty
- To Slack
- To Telegram

- Via Webhook
- Through any other channels supported in Alertmanager

Examples of DKP monitoring integration with external systems are available in Configuring integrations.

Cluster SLA monitoring

Availability assessment in DKP is performed by the `upmeter` module.

Composition of the `upmeter` module:

- **agent**: Runs on master nodes and performs availability probes, sends results to the server.
- **upmeter**: Collects results and maintains an API server for their retrieval.
- **front**:
  - **status**: Shows availability level for the last 10 minutes (requires authorization, but it can be disabled).
  - **webui**: Shows a dashboard with statistics on probes and availability groups (requires authorization).
- **smoke-mini**: Maintains continuous *smoke testing* using StatefulSet.

The module sends about 100 metric readings every 5 minutes. This value depends on the number of enabled Deckhouse Kubernetes Platform modules.

Additional resources

- The list of alerts
- Integration with external monitoring systems
- Web interface of the monitoring
- Web interface of the platform
- Module `prometheus` documentation

Configuring network interaction and cluster node monitoring

Network interaction monitoring

DKP can perform monitoring of network interaction between all cluster nodes, as well as between cluster nodes and external hosts. When monitoring is configured, each node sends ICMP packets twice per second to all other cluster nodes (and to optional external nodes) and exports data to the monitoring system.

Features of network interaction monitoring:

- Automatically checks the availability of all cluster nodes (and, optionally, external systems) using ICMP (ping) — testing is started every two seconds.
- All results are exported in metrics format to the Prometheus monitoring system.
- Included is a ready-made dashboard for Grafana, where current availability, delay schedules, and potential network connectivity issues are visualized in real time (for details, refer to Grafana).
- Allows you to quickly identify nodes with degraded connectivity and speeds up the response to incidents.

The `monitoring-ping` module tracks any changes to the `.status.addresses` field of a node. If changes are detected, a hook is triggered that collects the complete list of node names and their addresses, and passes it to the DaemonSet, which recreates the pods. Thus, `ping` checks always the current list of nodes.

> ⚠ The `monitoring-ping` module must be enabled.

Adding additional IP addresses for monitoring

To add additional monitoring IP addresses, use the `externalTargets` parameter of the `monitoring-ping` module.

Example module configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-ping
spec:
  version: 1
  enabled: true
  settings:
    externalTargets:
    - name: google-primary
      host: 8.8.8.8
    - name: yaru
      host: ya.ru
    - host: youtube.com
```

> **i** The `name` field is used in Grafana to display related data. If the `name` field is not specified, the required `host` field is used.

Cluster node monitoring

To enable cluster node monitoring, you need to enable the `monitoring-kubernetes` module if it's not already enabled. You can enable cluster monitoring in the Deckhouse web interface, or using the following command:

```
d8 system module enable monitoring-kubernetes
```

In a similar way, you can enable the `monitoring-kubernetes-control-plane` and `extended-monitoring` modules.

Additional resources

- Module `console` documentation
- Module `extended-monitoring` documentation
- Module `monitoring-kubernetes` documentation
- Module `monitoring-kubernetes-control-plane` documentation
- Module `monitoring-ping` documentation

Writing Prometheus data to longterm storage

Prometheus supports the remote_write mechanism for sending data from a local Prometheus instance to a separate long-term storage (for example, VictoriaMetrics). In DKP, support for this mechanism is implemented using the PrometheusRemoteWrite custom resource.

> **i** For VictoriaMetrics, detailed information about ways to send data to vmagent can be found in the VictoriaMetrics documentation.

Example of minimal PrometheusRemoteWrite

Below is an example of minimal configuration for the PrometheusRemoteWrite custom resource:

```
apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: test-remote-write
spec:
  url: https://victoriametrics-test.domain.com/api/v1/write
```

Example of advanced PrometheusRemoteWrite

Below is an example of advanced configuration for the PrometheusRemoteWrite custom resource:

```
apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: test-remote-write
spec:
  url: https://victoriametrics-test.domain.com/api/v1/write
  basicAuth:
    username: username
    password: password
  writeRelabelConfigs:
  - sourceLabels: [__name__]
    action: keep
    regex: prometheus_build_.*|my_cool_app_metrics_.*
  - sourceLabels: [__name__]
    action: drop
    regex: my_cool_app_metrics_with_sensitive_data
```

For a complete description of the custom resource fields, refer to the prometheus module documentation.

Additional resources

- Module `prometheus` custom resources

External access configuration

Connecting Prometheus to Third-Party Grafana

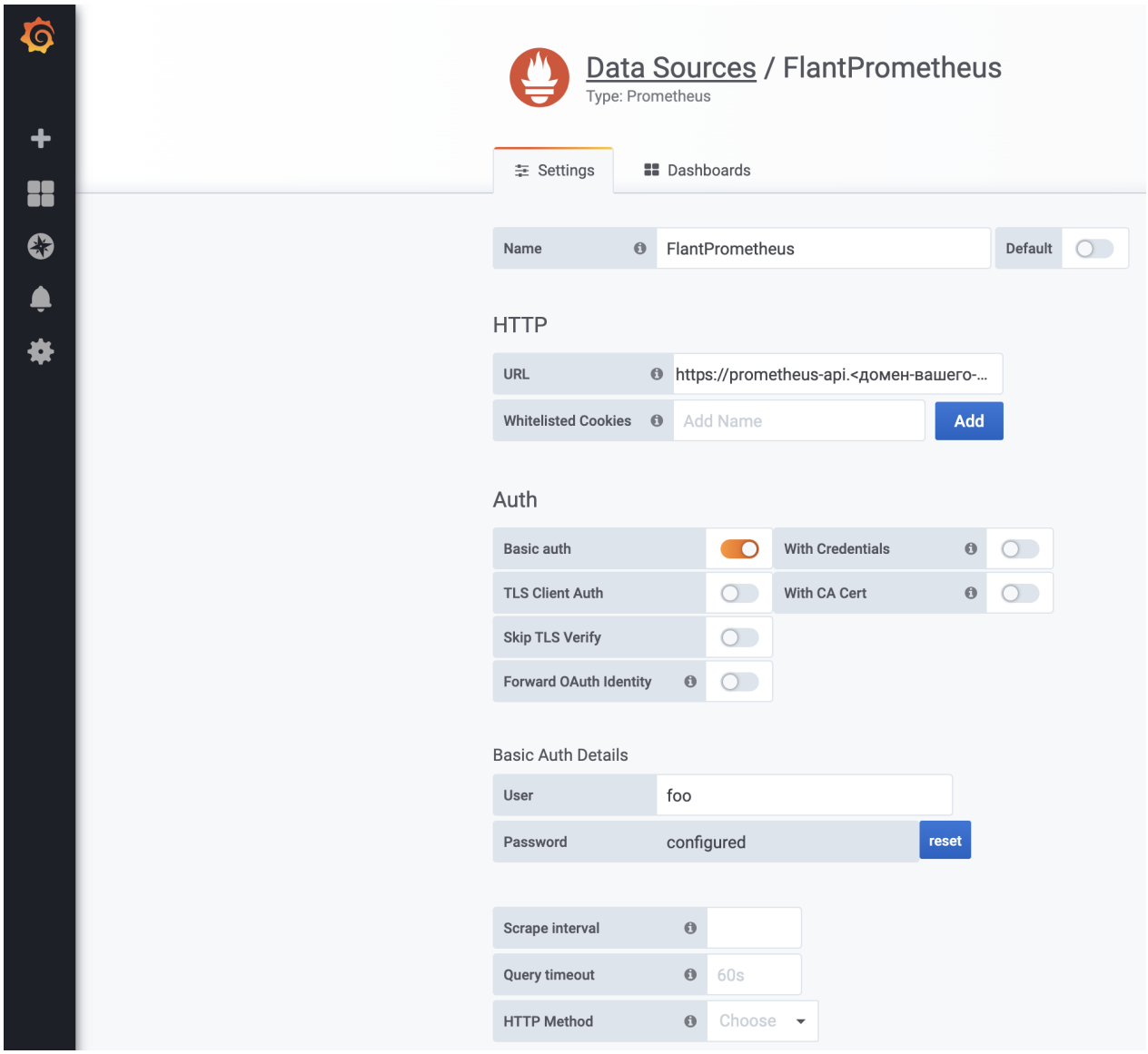
Each `ingress-nginx-controller` has certificates that, when specified as client certificates, will allow connection to Prometheus. All you need to do is create an additional `Ingress` resource.

> **i**
>
> The example below assumes that the Secret `example-com-tls` already exists in the d8-monitoring namespace.
>
> The names for Ingress `my-prometheus-api` and Secret `my-basic-auth-secret` are provided as examples. Replace them with more suitable ones for your setup.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-prometheus-api
  namespace: d8-monitoring
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
    nginx.ingress.kubernetes.io/auth-type: basic
    nginx.ingress.kubernetes.io/auth-secret: my-basic-auth-secret
    nginx.ingress.kubernetes.io/app-root: /graph
    nginx.ingress.kubernetes.io/configuration-snippet: |
      proxy_ssl_certificate /etc/nginx/ssl/client.crt;
      proxy_ssl_certificate_key /etc/nginx/ssl/client.key;
      proxy_ssl_protocols TLSv1.2;
      proxy_ssl_session_reuse on;
spec:
  ingressClassName: nginx
  rules:
  - host: prometheus-api.example.com
    http:
      paths:
      - backend:
          service:
            name: prometheus
            port:
              name: https
        path: /
        pathType: ImplementationSpecific
  tls:
  - hosts:
    - prometheus-api.example.com
    secretName: example-com-tls
---
apiVersion: v1
kind: Secret
metadata:
  name: my-basic-auth-secret
  namespace: d8-monitoring
type: Opaque
data:
  # Basic-auth string is hashed using htpasswd.
  auth: Zm9vOiRhcHIxJE9GRzNYeWJwJGNrTDBGSERBa29YWUlsSDkuY3lzVDAK  # foo:bar
```

Add the data source to Grafana:

> **i** As the URL, you need to specify `https://prometheus-api.<your-cluster-domain>`.



- **Basic authorization** is not a reliable security measure. It is recommended to implement additional security measures, such as specifying the `nginx.ingress.kubernetes.io/whitelist-source-range` annotation.
- Due to the need to create an Ingress resource in the system namespace, this connection method is **not recommended**. DKP **does not guarantee** the preservation of this connection scheme's functionality due to its active continuous updates.
- This Ingress resource can be used to access the Prometheus API not only for Grafana but also for other integrations, such as Prometheus federation.

Connecting Third-Party Applications to Prometheus

The connection to Prometheus is secured using kube-rbac-proxy. To connect, create a `ServiceAccount` with the necessary permissions.

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app:prometheus-access
rules:
- apiGroups: ["monitoring.coreos.com"]
  resources: ["prometheuses/http"]
  resourceNames: ["main", "longterm"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: app:prometheus-access
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app:prometheus-access
subjects:
- kind: ServiceAccount
  name: app
  namespace: default
```

Execute the request using the `curl` command:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: app-curl
  namespace: default
spec:
  template:
    metadata:
      name: app-curl
    spec:
      serviceAccountName: app
      containers:
      - name: app-curl
        image: curlimages/curl:7.69.1
        command: ["sh", "-c"]
        args:
        - >-
          curl -H "Authorization: Bearer $(cat
/var/run/secrets/kubernetes.io/serviceaccount/token)" -k -f
          https://prometheus.d8-monitoring:9090/api/v1/query_range?
query=up\&start=1584001500\&end=1584023100\&step=30
      restartPolicy: Never
  backoffLimit: 4
```

The `Job` should complete successfully.

Metrics Collection via Gateway (Pushgateway)

Prometheus, which is the foundation of the DKP monitoring system, primarily uses a pull model for

metrics collection. In this approach, DKP polls metric exporters. When applying the pull model is difficult, for example, for services without a permanent network interface, you can use metrics collection via a gateway (Pushgateway). Pushgateway allows such tasks to send metrics themselves, which can then be collected by Prometheus. It's important to note that Pushgateway can become a single point of failure and bottleneck in the system. How to send metrics from an application to Pushgateway can be learned from the Prometheus documentation.

Example of configuring metrics collection via gateway (Pushgateway):

- Enable and configure the `prometheus-pushgateway` module.

  You can enable the module in the web interface (Deckhouse Console), or using the following command:

  ```
  d8 system module enable prometheus-pushgateway
  ```

- Specify the gateway names in the `instances` parameter of the `prometheus-pushgateway` module through the web interface, or using the following command:

  ```
  d8 k edit mc prometheus-pushgateway
  ```

  Example module configuration:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: prometheus-pushgateway
  spec:
    version: 1
    enabled: true
    settings:
      instances:
      - first
      - second
      - another
  ```

  The address of the PushGateway instance named `first` from a pod container will be: `http://first.kube-prometheus-pushgateway:9091`.

- Check metrics sending.

  Example of sending metrics via curl:

  ```
  echo "test_metric{env="dev"} 3.14" | curl --data-binary @- http://first.kube-prometheus-pushgateway:9091/metrics/job/myapp
  ```

- Check that the metric appeared in the monitoring system. It will be available 30 seconds after data collection.

  Example PromQL query:

  ```
  test_metric{container="prometheus-pushgateway", env="dev", exported_job="myapp",
      instance="10.244.1.155:9091", job="prometheus-pushgateway", pushgateway="prometheus-pushgateway", tier="cluster"} 3.14
  ```

Removing Metrics from Gateway (Pushgateway)

Example of removing all metrics from group `{instance="10.244.1.155:9091",job="myapp"}` via curl:

```
curl -X DELETE http://first.kube-prometheus-pushgateway:9091/metrics/job/myapp/instance/10.244.1.155:9091
```

Note that PushGateway stores received metrics in memory. When the PushGateway pod restarts, all metrics will be lost.

Additional resources

- Module `prometheus-pushgateway` documentation

Configuring a system for collecting and storing metrics

> **i** Starting from version 1.71, Deckhouse Kubernetes Platform uses Deckhouse Prom++ instead of Prometheus.

What does Prometheus do?

Prometheus collects metrics and executes rules:

- For each *target* (monitoring target) at a specified interval `scrape_interval`, Prometheus makes an HTTP request to this *target*, receives metrics in its own format in response, and stores them in its database.
- Every `evaluation_interval` it processes rules (*rules*), based on which:
  - it sends alerts;
  - or stores new metrics (result of rule execution) in its database.

How does Prometheus work?

Prometheus is installed by the `prometheus` module of DKP, which performs the following functions:

- Defines the following custom resources:
  - `Prometheus`: Defines the *Prometheus* installation (cluster).
  - `ServiceMonitor`: Defines how to collect metrics from services.
  - `Alertmanager`: Defines the *Alertmanager* cluster.
  - `PrometheusRule`: Defines the list of *Prometheus rules*.
- Monitors these resources, and also:
  - Generates `StatefulSet` with *Prometheus* itself.
  - Creates secrets with configuration files necessary for Prometheus operation (`prometheus.yaml` — Prometheus configuration, and `configmaps.json` — configuration for `prometheus-config-reloader`).
  - Monitors `ServiceMonitor` and `PrometheusRule` resources and updates *Prometheus* configuration files by modifying secrets based on them.

The module can be enabled using the following ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  settings:
    auth:
      password: xxxxxx
    retentionDays: 7
    storageClass: rbd
    nodeSelector:
      node-role/monitoring: ""
    tolerations:
    - key: dedicated.deckhouse.io
      operator: Equal
      value: monitoring
```

A complete description of all settings is available in the prometheus module documentation.

Additional resources

- Module `prometheus` documentation

Cluster SLA monitoring

DKP can collect statistics about the availability of cluster components and DKP components themselves. This data allows evaluating SLA compliance and provides availability information in the web interface.

Additionally, using the UpmeterRemoteWrite custom resource, you can export availability metrics via the Prometheus Remote Write protocol.

To start collecting availability metrics and activate the interface, enable the `upmeter` module in the Deckhouse web interface or use the following command:

```
d8 system module enable upmeter
```

Module configuration

The `upmeter` module is configured using the `upmeter` ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: upmeter
spec:
  version: 3
  enabled: true
  settings:
```

A complete list of all settings is available in the module documentation.

Interface

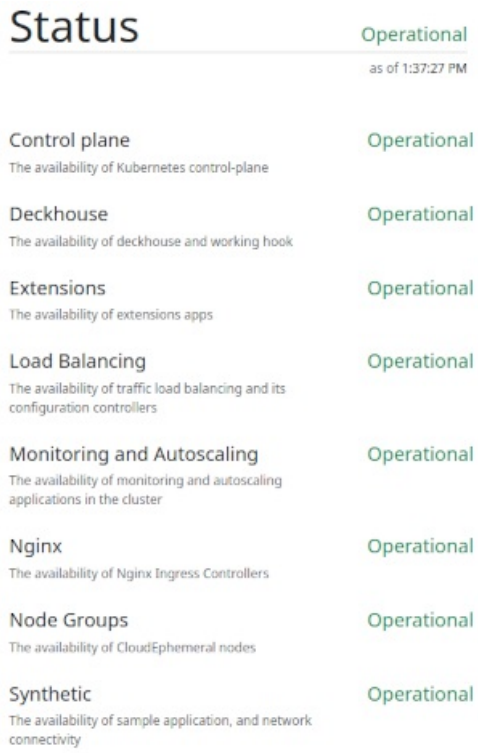
DKP provides two web interfaces for availability assessment:

- Status page.

  You can get the page address in the web interface on the main page in the “Tools” section (Status page block), or by running the command:

  ```
  d8 k -n d8-upmeter get ing status -o jsonpath='{.spec.rules[*].host}'
  ```

  Example of the status page web interface:

- Component availability page.

You can get the page address in the web interface on the main page in the “Tools” section (Component availability block), or by running the command:

```
d8 k -n d8-upmeter get ing webui -o jsonpath='{.spec.rules[*].host}'
```

Example of the component availability page:



Status metrics export

Example of UpmeterRemoteWrite configuration for exporting status metrics via the Prometheus Remote Write protocol:

```
apiVersion: deckhouse.io/v1
kind: UpmeterRemoteWrite
metadata:
  labels:
    heritage: upmeter
    module: upmeter
  name: victoriametrics
spec:
  additionalLabels:
    cluster: cluster-name
    some: fun
  config:
    url: https://upmeter-victoriametrics.whatever/api/v1/write
    basicAuth:
      password: "Cdp#Cd.OxfZsx4*89SZ"
      username: upmeter
  intervalSeconds: 300
```

Authentication

By default, the `user-authn` module is used for authentication. You can also configure authentication via `externalAuthentication`. If these options are disabled, the module will enable basic authentication with a generated password.

You can view the generated password using the following command:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values upmeter
-o json | jq '.upmeter.internal.auth.webui.password'
```

To generate a new password, you need to delete the Secret:

```
d8 k -n d8-upmeter delete secret/basic-auth-webui
```

You can view the generated password for the status page using the following command:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values upmeter
-o json | jq '.upmeter.internal.auth.status.password'
```

To generate a new password for the status page, you need to delete the secret:

```
d8 k -n d8-upmeter delete secret/basic-auth-status
```

> **Attention!** The `auth.status.password` and `auth.webui.password` parameters are no longer supported.

Behavior of upmeter pods

Upmeter tests create temporary pods to check that Kubernetes components are working. As a result, some pods are periodically deleted, remain in the `Pending` state, or move between nodes.

The following objects take part in the checks:

- `upmeter-probe-scheduler`: Checks the scheduler. The test creates a pod, schedules it to a node, and then deletes it.
- `upmeter-probe-controller-manager`: Checks `kube-controller-manager`. The test creates a StatefulSet and verifies that the StatefulSet created a pod. This test does not check pod placement on a node, so it creates a pod that cannot be scheduled and stays in the `Pending` state. Then the StatefulSet is deleted, and the test verifies that the spawned pod is deleted as well.
- `smoke-mini`: Checks network connectivity between nodes. Five StatefulSets with one replica each are created. The test checks connectivity between `smoke-mini` pods and `upmeter-agent` pods on master nodes. Once a minute, one of the `smoke-mini` pods is moved to another node.

Additional resources

- Module `console` documentation
- Module `upmeter` documentation
- Module `user-authn` documentation

Configuring integrations

Redirecting alerts to Zabbix

Deckhouse Kubernetes Platform supports integration with the Zabbix monitoring system. For this purpose, an external script is used that receives alerts from Deckhouse via `kubectl` and sends them to Zabbix using the Zabbix agent.

The script requires:

- installed and configured zabbix-agent;
- utilities `bash`, `jq`;
- `kubectl` with a working configuration file and permissions to execute the `kubectl get clusteralerts` command.

Installation

1. Import the template into Zabbix:
   - In the Zabbix web interface, go to the “Data collection → Templates” section
   - Click “Import” and upload the `zbx_export_templates.yaml` file:

```
zabbix_export:
  version: '6.4'
  template_groups:
    - uuid: 7df96b18c230490a9a0a9e2307226338
      name: Templates
  templates:
    - uuid: 91d9cf9d023749a3997bc6429ef45fe1
      template: 'Module d8alerts'
      name: 'Deckhouse Alerts'
      groups:
        - name: Templates
      discovery_rules:
        - uuid: e9be18e9305e48e887c43e1ad9ea9c31
```

```
        name: 'Deckhouse Alert Discovery'
        key: d8alerts.discovery
        lifetime: 3d
        item_prototypes:
          - uuid: c4152d379d7a46cab23f7437cdb632c1
            name: 'Deckhouse alert: {#ALERTNAME} severity'
            key: 'd8alerts.severity[{#ALERTID}]'
            history: 1d
            trends: '0'
            description: |
              Summary: {#SUMMARY}

              {#DESCRIPTION}

              Labels: {#LABELS}
              To get more information: `kubectl describe clusteralert
{#ALERTID}`
            preprocessing:
              - type: REGEX
                parameters:
                  - (\d+)
                  - \1
            tags:
              - tag: Application
                value: deckhouse
            trigger_prototypes:
              - uuid: b27f8d03dc4c4d33bab4193a6ce57ae7
                expression: 'last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) <= 2 and last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) > 0'
                name: '{#ALERTNAME} is firing'
                priority: DISASTER
                description: |
                  Summary: {#SUMMARY}

                  {#DESCRIPTION}

                  Labels: {#LABELS}
                  To get more information: `kubectl describe clusteralert
{#ALERTID}`
              - uuid: d1935044cf3f49df8bc53f738e36b683
                expression: 'last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) >= 3 and last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) <= 4'
                name: '{#ALERTNAME} is firing'
                priority: HIGH
                description: |
                  Summary: {#SUMMARY}

                  {#DESCRIPTION}

                  Labels: {#LABELS}
                  To get more information: `kubectl describe clusteralert
{#ALERTID}`
              - uuid: 0b134c599d9d4de79f906fbd8e749ec2
                expression: 'last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) >= 5 and last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) <= 6'
                name: '{#ALERTNAME} is firing'
                priority: AVERAGE
                description: |
                  Summary: {#SUMMARY}

                  {#DESCRIPTION}
```

```
                Labels: {#LABELS}
                To get more information: `kubectl describe clusteralert
{#ALERTID}`
            - uuid: 6967dc80d6414239b2d597447815048a
              expression: 'last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) >= 7 and last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) <= 8'
              name: '{#ALERTNAME} is firing'
              priority: WARNING
              description: |
                Summary: {#SUMMARY}

                {#DESCRIPTION}

                Labels: {#LABELS}
                To get more information: `kubectl describe clusteralert
{#ALERTID}`
            - uuid: 8e531a2e6a0f47849c1dffea9a1733e1
              expression: 'last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) >= 9 and last(/Module
d8alerts/d8alerts.severity[{#ALERTID}]) <= 10'
              name: '{#ALERTNAME} is firing'
              priority: INFO
              description: |
                Summary: {#SUMMARY}

                {#DESCRIPTION}

                Labels: {#LABELS}
                To get more information: `kubectl describe clusteralert
{#ALERTID}`
```

- After installing the template, assign it to the group corresponding to the agents from which metrics will be collected.

2. Configure the Zabbix agent:

   - Copy the `d8alerts.conf` file to the directory specified in the `Include` parameter of the main Zabbix agent config (usually located at `/etc/zabbix/zabbix_agentd.d/`):

     ```
     # LLD of deckhouse cluster alerts
     UserParameter=d8alerts.discovery,/etc/zabbix/scripts/clusteralerts.sh
     discovery

     # Severity of a specific alert by its ID
     UserParameter=d8alerts.severity[*],/etc/zabbix/scripts/clusteralerts.sh
     severity "$1"
     ```

   - Copy the `clusteralerts.sh` script to the `/etc/zabbix/scripts/` directory:

```
#!/bin/bash

MODE="$1"
ALERT_ID="$2"

get_alerts_json() {
  kubectl get clusteralerts -o json
}

get_alert_by_id() {
  local id="$1"
  kubectl get clusteralerts "$id" -o json
}

if [[ "$MODE" == "discovery" ]]; then
  get_alerts_json | jq -c '{
      data: [.items[] | {
        "{#ALERTID}": .metadata.name,
        "{#ALERTNAME}": .alert.name,
        "{#SEVERITY}": (.alert.severityLevel | tonumber),
        "{#DESCRIPTION}": .alert.description,
        "{#SUMMARY}": .alert.summary,
        "{#LABELS}": (.alert.labels | to_entries | map("\(.key)=\(.value)") |
join(","))
      }]
    }'

elif [[ "$MODE" == "severity" && -n "$ALERT_ID" ]]; then
  get_alert_by_id "$ALERT_ID" | jq -r '.alert.severityLevel'

else
  echo "Invalid usage"
  exit 1
fi
```

- Make sure the script has execute permissions:

```
chmod +x /etc/zabbix/scripts/clusteralerts.sh
```

3. Check access:
   - Ensure the script has access to the cluster and can retrieve alert information:

     ```
     /etc/zabbix/scripts/clusteralerts.sh discovery
     ```

     The output should contain a list of alerts with their statuses and severity levels.
4. Restart the Zabbix agent to apply changes:

   ```
   systemctl restart zabbix-agent
   ```

Troubleshooting

1. Check Zabbix agent logs:

   ```
   tail -f /var/log/zabbix/zabbix_agentd.log
   ```

2. Check script operation:

   ```
   /etc/zabbix/scripts/clusteralerts.sh discovery
   /etc/zabbix/scripts/clusteralerts.sh severity "ALERT_ID"
   ```

Make sure the script executes correctly and returns expected data.

3. Check the `zabbix` user permissions. Ensure the agent runs the script as a user with the necessary cluster access rights:
   - The script runs as the `zabbix` user:

     ```
     sudo -u zabbix /etc/zabbix/scripts/clusteralerts.sh
     ```

   - The `KUBECONFIG` variable is available to the user:

     If the Kubernetes configuration file is not available by default, specify it explicitly. To do this, save the kubeconfig, for example, in `/etc/zabbix/kubeconfig` and add to the agent configuration:

     ```
     UserParameter=d8alerts.discovery,export
     KUBECONFIG=/etc/zabbix/kubeconfig;/etc/zabbix/scripts/clusteralerts.sh discovery
     ```

Sending alerts to Telegram

Alertmanager supports direct sending of alerts to Telegram.

Create a Secret in the `d8-monitoring` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: telegram-bot-secret
  namespace: d8-monitoring
stringData:
  token: "562696849:AAExcuJ8H6z4pTlPuocbrXXXXXXXXXXx"
```

Deploy the CustomAlertManager custom resource:

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: telegram
spec:
  type: Internal
  internal:
    receivers:
      - name: telegram
        telegramConfigs:
          - botToken:
              name: telegram-bot-secret
              key: token
            chatID: -30490XXXXX
    route:
      groupBy:
        - job
      groupInterval: 5m
      groupWait: 30s
      receiver: telegram
      repeatInterval: 12h
```

The `token` field in the Secret and `chatID` in the `CustomAlertmanager` resource need to be set to your own values. More details about the Telegram API.

Example of sending alerts to Slack with filter

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: slack
spec:
  internal:
    receivers:
    - name: devnull
    - name: slack
      slackConfigs:
      - apiURL:
          key: apiURL
          name: slack-apiurl
        channel: {{ dig .Values.werf.env .Values.slack.channel._default
.Values.slack.channel }}
        fields:
        - short: true
          title: Severity
          value: '{{`{{  .CommonLabels.severity_level }}`}}'
        - short: true
          title: Status
          value: '{{`{{ .Status }}`}}'
        - title: Summary
          value: '{{`{{ range .Alerts }}`}}{{`{{ .Annotations.summary }}`}} {{`{{ end
}}`}}'
        - title: Description
          value: '{{`{{ range .Alerts }}`}}{{`{{ .Annotations.description }}`}} {{`{{
end }}`}}'
        - title: Labels
          value: '{{`{{ range .Alerts }}`}} {{`{{ range .Labels.SortedPairs }}`}}{{`{{
printf "%s:
            %s\n" .Name .Value }}`}}{{`{{ end }}`}}{{`{{ end }}`}}'
        - title: Links
          value: '{{`{{ (index .Alerts 0).GeneratorURL }}`}}'
        title: '{{`{{ .CommonLabels.alertname }}`}}'
    route:
      groupBy:
      - '...'
      receiver: devnull
      routes:
        - matchers:
          - matchType: =~
            name: severity_level
            value: "^[4-9]$"
          receiver: slack
      repeatInterval: 12h
  type: Internal
```

Example of sending alerts to Opsgenie

```
- name: opsgenie
      opsgenieConfigs:
        - apiKey:
            key: data
            name: opsgenie
          description: |
            {{ range .Alerts }}{{ .Annotations.summary }} {{ end }}
            {{ range .Alerts }}{{ .Annotations.description }} {{ end }}
          message: '{{ .CommonLabels.alertname }}'
          priority: P1
          responders:
            - id: team_id
              type: team
```

Example of sending alerts via email

Create a secret with the email account password. Specify the password encoded in Base64 format in the `password` field:

```
apiVersion: v1
kind: Secret
metadata:
  name: am-mail-server-pass
  namespace: d8-monitoring
data:
  password: BASE64_ENCODED_PASSWORD_HERE
```

Modify the values in the CustomAlertManager resource example according to the values relevant to your infrastructure and apply it:

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: mail
spec:
  type: Internal
  internal:
    receivers:
      - name: devnull
      - name: mail
        emailConfigs:
          - to: oncall@example.com
            from: prom@example.com
            smarthost: mx.example.com:587
            authIdentity: prom@example.com
            authUsername: prom@example.com
            authPassword:
              key: password
              name: am-mail-server-pass
            # If you use a custom CA on the server, you can place the public part of
the CA in a ConfigMap in the d8-monitoring namespace
            # tlsConfig:
            #   insecureSkipVerify: true
            #   ca:
            #     configMap:
            #       key: ca.pem
            #       name: alertmanager-mail-server-ca
            sendResolved: true
            requireTLS: true
    route:
      groupBy:
        - job
      groupInterval: 5m
      groupWait: 30s
      receiver: devnull
      repeatInterval: 24h
      routes:
        - matchers:
          - matchType: =~
            name: severity_level
            value: "^[1-4]$"
          receiver: mail
```

Additional resources

- Module `prometheus` custom resources

Logging

Deckhouse Kubernetes Platform (DKP) provides built-in tools for log collection, delivery, and short-term storage.

DKP logging capabilities:

- Collect logs from cluster pods and nodes.
- Process logs, including metadata enrichment and message filtering.
- Deliver to various storage and analysis systems, including Loki, Elasticsearch, Splunk, and others.
- Short-term log storage in the cluster with search and visualization capabilities through Grafana.

The following sections describe how to configure log collection and delivery, as well as organize their short-term storage within the cluster.

Log collection and delivery

Deckhouse Kubernetes Platform (DKP) provides log collection and delivery from cluster nodes and pods to internal or external storage systems.

DKP allows you to:

- Collect logs from all or specific pods and namespaces.
- Filter logs by labels, message content, and other criteria.
- Send logs to multiple storage systems simultaneously (e.g., Loki and Elasticsearch).
- Enrich logs with Kubernetes metadata.
- Use log buffering to improve performance.

The general mechanism of log collection, delivery, and filtering is described in detail in the Architecture section.

DKP administrators can configure log collection and delivery using three custom resources:

- ClusterLoggingConfig: Describes log sources at the cluster level, including collection, filtering, and parsing rules.
- PodLoggingConfig: Describes log sources within a specified namespace, including collection, filtering, and parsing rules.
- ClusterLogDestination: Defines log storage parameters.

Based on these resources, a *pipeline* is formed, which is used in DKP to read logs and further work with them using the `log-shipper` module. A complete list of `log-shipper` module settings is available in the separate documentation section.

Configuring log collection and delivery

Below is a basic DKP configuration option where logs from all cluster pods are sent to Elasticsearch-based storage.

To configure, follow these steps:

1. Enable the `log-shipper` module using the following command:

   ```
   d8 system module enable log-shipper
   ```

2. Create a ClusterLoggingConfig resource that defines log collection rules. This resource allows you to configure log collection from pods in a specific namespace and with specific labels, flexibly configure multi-line log parsing, and set other rules.

   In this example, it is specified that logs should be collected from all pods and sent to Elasticsearch:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ClusterLoggingConfig
   metadata:
     name: all-logs
   spec:
     type: KubernetesPods
     destinationRefs:
     - es-storage
   ```

3. Create a ClusterLogDestination resource, which describes the parameters for sending logs to storage. This resource allows you to specify one or more storage systems and describe connection parameters, buffering, and additional labels that will be applied to logs before sending.

   In this example, Elasticsearch is specified as the receiving storage:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: logs-%F
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmV0IC1uCg==
```

Integration with external systems

You can configure DKP to work with external log storage and analysis systems, such as Elasticsearch, Splunk, Logstash, and others, using the `type` parameter of the ClusterLogDestination resource.

Elasticsearch

To send logs to Elasticsearch, create a ClusterLogDestination resource following this example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: logs-%F
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmV0IC1uCg==
```

Using index templates

To send messages to specific indices based on metadata using index templates, use the following configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: "k8s--%F"
```

In the example above, a separate index will be created in Elasticsearch for each Kubernetes namespace.

This feature is useful in combination with the `extraLabels` parameter:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: "k8s---%F"
  extraLabels:
    service: ""
```

- If the message has JSON format, the `service_name` field of this JSON document is moved to the metadata level.
- The new metadata field `service` is used in the index template.

Support for Elasticsearch < 6.X

To work with Elasticsearch versions prior to 6.0, enable support for `docType` indices using the ClusterLogDestination resource:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    docType: "myDocType" # Specify the value here. It should not start with '_'.
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmV0IC1uCg==
```

Splunk

To configure sending events to Splunk, follow these steps:

1. Configure Splunk:
   - Define the endpoint. It should match your Splunk instance name with port `8088`, but without specifying the path, for example, `https://prd-p-xxxxxx.splunkcloud.com:8088`.
   - Create an access token. To do this, in Splunk, open the **Setting** -> **Data inputs** section, add a new **HTTP Event Collector** and copy the generated token.
   - Specify the Splunk index for storing logs, for example, `logs`.
2. Configure DKP by adding a ClusterLogDestination resource to send logs to Splunk:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: splunk
spec:
  type: Splunk
  splunk:
    endpoint: https://prd-p-xxxxxx.splunkcloud.com:8088
    token: xxxx-xxxx-xxxx
    index: logs
    tls:
      verifyCertificate: false
      verifyHostname: false
```

> **i** `destination` does not support pod labels for indexing. To add the required labels, use the `extraLabels` option:
>
> ```
> extraLabels:
>  pod_label_app: ''
> ```

Logstash

To configure sending logs to Logstash, do the following:

1. Configure an incoming `tcp` stream with `json` codec on the Logstash side.

   Example Logstash configuration:

   ```
   input {
     tcp {
       port => 12345
       codec => json
     }
   }
   output {
     stdout { codec => json }
   }
   ```

2. Add a ClusterLogDestination resource:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ClusterLogDestination
   metadata:
     name: logstash
   spec:
     type: Logstash
     logstash:
       endpoint: logstash.default:12345
   ```

Graylog

To configure sending logs to Graylog, do the following:

1. Ensure that Graylog has an incoming stream configured to receive messages via TCP protocol on the specified port.
2. Create a ClusterLogDestination resource following the example:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ClusterLogDestination
   metadata:
     name: test-socket2-dest
   spec:
     type: Socket
     socket:
       address: graylog.svc.cluster.local:9200
       mode: TCP
       encoding:
         codec: GELF
   ```

Message formats

You can choose the format of sent messages using the `.encoding.codec` parameter of the ClusterLogDestination resource:

- CEF

- GELF
- JSON
- Syslog
- Text

Below are configuration examples for some of them.

Syslog

Use the following configuration example to send messages via socket using TCP protocol in syslog format:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: rsyslog
spec:
  type: Socket
  socket:
    mode: TCP
    address: 192.168.0.1:3000
    encoding:
      codec: Syslog
  extraLabels:
    syslog.severity: "alert"
    # The request_id field must be present in the message.
    syslog.message_id: ""
```

CEF

DKP can send logs in CEF format by using `codec: CEF`, with overriding `cef.name` and `cef.severity` based on values from the `message` field of the application log in JSON format.

In the example below, `app` and `log_level` are keys containing values for overriding:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: siem-kafka
spec:
  extraLabels:
    cef.name: ''
    cef.severity: ''
  type: Kafka
  kafka:
    bootstrapServers:
      - my-cluster-kafka-brokers.kafka:9092
    encoding:
      codec: CEF
    tls:
      verifyCertificate: false
      verifyHostname: true
    topic: logs
```

You can also set values manually:

```
extraLabels:
  cef.name: 'TestName'
  cef.severity: '1'
```

Log transformation

You can configure one or more types of transformations that will be applied to logs before sending to

storage.

Converting records to structured objects

The `ParseMessage` transformation allows you to convert a string in the `message` field to a structured JSON object based on one or more specified formats (String, Klog, SysLog, and others).

> ⚠ When using multiple `ParseMessage` transformations, string conversion (`sourceFormat: String`) must be performed last.

Example configuration for converting mixed format records:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLogDestination
metadata:
  name: parse-json
spec:
  ...
  transformations:
  - action: ParseMessage
    parseMessage:
      sourceFormat: JSON
  - action: ParseMessage
    parseMessage:
      sourceFormat: Klog
  - action: ParseMessage
    parseMessage:
      sourceFormat: String
      string:
        targetField: "text"
```

Example of the original log record:

```
/docker-entrypoint.sh: Configuration complete; ready for start up
{"level" : { "severity": "info" },"msg" : "fetching.module.release"}
I0505 17:59:40.692994   28133 klog.go:70] hello from klog
```

Transformation result:

```
{... "message": {
  "text": "/docker-entrypoint.sh: Configuration complete; ready for start up"
  }
}
{... "message": {
  "level" : "{ "severity": "info" }",
  "msg" : "fetching.module.release"
  }
}
{... "message": {
  "file":"klog.go",
  "id":28133,
  "level":"info",
  "line":70,
  "message":"hello from klog",
  "timestamp":"2025-05-05T17:59:40.692994Z"
  }
}
```

Label replacement

The `ReplaceKeys` transformation allows you to recursively replace all matches of the `source` pattern

with the `target` value in the specified label keys.

> ⚠ Before applying the `ReplaceKeys` transformation to the `message` field or its nested fields, convert the log record to a structured object using the `ParseMessage` transformation.

Example configuration for replacing dots with underscores in labels:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLogDestination
metadata:
  name: replace-dot
spec:
  ...
  transformations:
    - action: ReplaceKeys
      replaceKeys:
        source: "."
        target: "_"
        labels:
          - .pod_labels
```

Example of the original log record:

```
{"msg" : "fetching.module.release"} # Pod label pod.app=test
```

Transformation result:

```
{... "message": {
  "msg" : "fetching.module.release"
  },
  "pod_labels": {
    "pod_app": "test"
  }
}
```

Label removal

The `DropLabels` transformation allows you to remove specified labels from a structured JSON message.

> ⚠ Before applying the `DropLabels` transformation to the `message` field or its nested fields, convert the log record to a structured object using the `ParseMessage` transformation.

Example configuration with label removal and preliminary `ParseMessage` transformation:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLogDestination
metadata:
  name: drop-label
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: JSON
    - action: DropLabels
      dropLabels:
        labels:
          - .message.example
```

Example of the original log record:

```
{"msg" : "fetching.module.release", "example": "test"}
```

Transformation result:

```
{... "message": {
  "msg" : "fetching.module.release"
  }
}
```

Log filtering

DKP provides filters to exclude unnecessary messages to optimize the log collection process:

- `labelFilter` — applied to metadata, such as container name (`container`), namespace (`namespace`), or pod name (`pod_name`);
- `logFilter` — applied to message fields, if the message is in JSON format.

Collecting logs from a specific container

To configure filtering using `labelFilter`, create a ClusterLoggingConfig resource, using the configuration below as an example.

In this case, the filter selects logs from containers named `nginx`, and then sends them to internal Loki-based storage.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: nginx-logs
spec:
  type: KubernetesPods
  labelFilter:
  - field: container
    operator: In
    values: [nginx]
  destinationRefs:
  - loki-storage
```

Collecting logs without a specific string

Example configuration for collecting logs with filtering via `labelFilter`, where the `NotRegex` operator excludes strings matching the specified regular expression.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: all-logs
spec:
  type: KubernetesPods
  destinationRefs:
  - loki-storage
  labelFilter:
  - field: message
    operator: NotRegex
    values:
    - .*GET /status" 200$
```

Kubelet audit events

Example configuration for collecting and filtering audit events related to kubelet operation, stored in the `/var/log/kube-audit/audit.log` file. Filtering is performed using `logFilter`, which searches for records in the `userAgent` field matching the regular expression `"kubelet.*"`.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
    - /var/log/kube-audit/audit.log
  logFilter:
  - field: userAgent
    operator: Regex
    values: ["kubelet.*"]
  destinationRefs:
  - loki-storage
```

DKP system logs

Example configuration for collecting DKP system logs located in the `/var/log/syslog` file. Message filtering using `labelFilter` allows you to select only those records that relate to the following components: `d8-kubelet-forker`, `containerd`, `bashible`, and `kernel`.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: system-logs
spec:
  type: File
  file:
    include:
    - /var/log/syslog
  labelFilter:
  - field: message
    operator: Regex
    values:
    - .*d8-kubelet-forker.*
    - .*containerd.*
    - .*bashible.*
    - .*kernel.*
  destinationRefs:
  - loki-storage
```

> **i** If you need logs from only one pod or a small group of pods, use `kubernetesPods` to limit the collection scope. Filters should be used only for fine-tuning.

Log buffering

Using buffering improves the reliability and performance of the log collection system. Buffering can be useful in the following cases:

- **Temporary connection issues**. If there are temporary interruptions or connection instability with the log storage system (e.g., Elasticsearch), the buffer allows temporarily storing logs and sending them when the connection is restored.
- **Load spike smoothing**. During sudden log volume spikes, the buffer helps smooth the peak load on the storage system, preventing its overload and potential data loss.
- **Performance optimization**. Buffering helps optimize the performance of the log collection system by accumulating logs and sending them in batches, which reduces the number of network requests and improves overall throughput.

The `buffer` parameter of the ClusterLogDestination resource is responsible for configuring buffering.

Example of enabling in-memory buffering

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    memory:
      maxEvents: 4096
    type: Memory
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

Example of enabling disk buffering

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    disk:
      maxSize: 1Gi
    type: Disk
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

Example of defining buffer overflow behavior

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    disk:
      maxSize: 1Gi
    type: Disk
    whenFull: DropNewest
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

Debugging and advanced features

Enabling debug logs for the log-shipper agent

To enable debug logs for the `log-shipper` agent on nodes with information about HTTP requests, connection reuse, tracing, and other data, enable the `debug` parameter in the `log-shipper` module configuration.

Example module configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: log-shipper
spec:
  version: 1
  enabled: true
  settings:
    debug: true
```

Additional information about log transmission channels

Using Vector commands, you can get additional information about data transmission channels.

First, connect to one of the `log-shipper` pods:

```
d8 k -n d8-log-shipper get pods -o wide | grep $node
d8 k -n d8-log-shipper exec $pod -it -c vector -- bash
```

Execute subsequent commands from the pod’s command shell.

Topology overview

To get a diagram of your configuration topology:

1. Run the `vector graph` command. A diagram in DOT format will be generated.
2. Use WebGraphviz or a similar service to render the diagram based on the DOT file content.

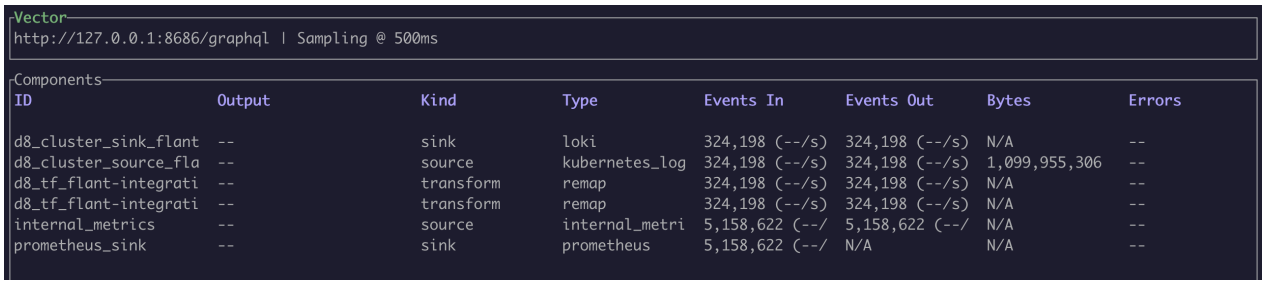
Example diagram for one log transmission channel in ASCII format:

```
+------------------------------------------------+
|  d8_cluster_source_flant-integration-d8-logs   |
+------------------------------------------------+
  |
  |
  v
+------------------------------------------------+
|       d8_tf_flant-integration-d8-logs_0        |
+------------------------------------------------+
  |
  |
  v
+------------------------------------------------+
|       d8_tf_flant-integration-d8-logs_1        |
+------------------------------------------------+
  |
  |
  v
+------------------------------------------------+
| d8_cluster_sink_flant-integration-loki-storage |
+------------------------------------------------+
```

Monitoring channel load

To view the traffic volume at each log processing stage, use the `vector top` command.

Example command output:



Getting raw and intermediate logs

To view input data at different log processing stages, use the `vector tap` command. By specifying the ID of a specific processing stage, you can see logs that arrive at that stage. Glob pattern selections are also supported, for example, `cluster_logging_config/*`.

Examples:

- Viewing logs before applying transformation rules (`cluster_logging_config/*` is the first processing stage according to the `vector graph` command output):

  ```
  vector tap 'cluster_logging_config/*'
  ```

- Modified logs entering the input of the next components in the channel chain:

  ```
  vector tap 'transform/*'
  ```

Debugging VRL rules

To debug rules in Vector Remap Language (VRL), use the `vector vrl` command.

Example VRL program:

```
. = {"test1": "lynx", "test2": "fox"}
del(.test2)
.
```

Adding support for new source or sink

The `log-shipper` module in DKP is built based on Vector with a limited set of cargo features, to minimize the size of the executable file and speed up the build.

To view the complete list of supported features, run the `vector list` command.

If the required source or sink is missing, add the corresponding cargo feature to the Dockerfile.

Special cases

Collecting logs from production namespaces using the labelSelector option

If your cluster namespaces are labeled (e.g., `environment=production`), you can use the `labelSelector` option to collect logs from production namespaces.

Example configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLoggingConfig
metadata:
  name: production-logs
spec:
  type: KubernetesPods
  kubernetesPods:
    namespaceSelector:
      labelSelector:
        matchLabels:
          environment: production
  destinationRefs:
  - loki-storage
```

Label for excluding pods and namespaces

DKP provides the `log-shipper.deckhouse.io/exclude=true` label for excluding specific pods and namespaces. It helps stop log collection from pods and namespaces without changing the global configuration.

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    log-shipper.deckhouse.io/exclude: "true"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-deployment
spec:
  ...
  template:
    metadata:
      labels:
        log-shipper.deckhouse.io/exclude: "true"
```

Additional resources

- Module `log-shipper` documentation

Short-term log storage

Deckhouse Kubernetes Platform (DKP) provides a built-in solution for short-term log storage based on the Grafana Loki project.

The storage is deployed in the cluster and integrated with the log collection system. After configuring ClusterLoggingConfig, PodLoggingConfig, and ClusterLogDestination resources, logs automatically flow

from all system components. The configured storage is added to Grafana as a data source for visualization and analysis.

Log collection from user applications is configured separately.

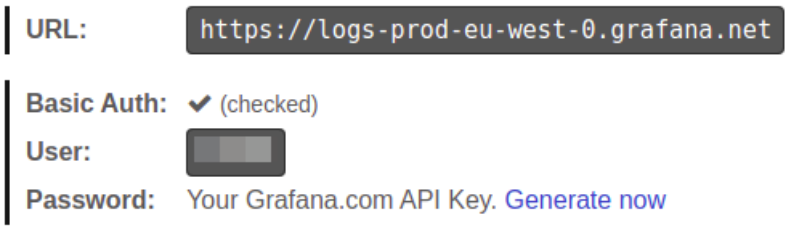
Short-term storage parameters are set in the `loki` module settings. You can configure disk size and retention period, specify the StorageClass to use, and set resource limits.

> ⚠ Short-term storage based on Grafana Loki does not support high availability mode. For long-term storage of important logs, use external storage.

Integration with Grafana Cloud

To configure DKP to work with the Grafana Cloud platform, follow these steps:

1. Create a Grafana Cloud API access key.
2. Encode the Grafana Cloud access token in Base64 format:




```
echo -n "<YOUR-GRAFANACLOUD-TOKEN>" | base64 -w0
```

3. Create a ClusterLogDestination resource following the example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  loki:
    auth:
      password: PFlPVVItR1JBRkFOQUNMT1VELVRPS0VOPg==
      strategy: Basic
      user: "<YOUR-GRAFANACLOUD-USERNAME>"
    endpoint: <YOUR-GRAFANACLOUD-URL> # For example https://logs-prod-us-
central1.grafana.net or https://logs-prod-eu-west-0.grafana.net
  type: Loki
```

Migration from Grafana Promtail

To migrate from Promtail, edit the Loki URL by removing the `/loki/api/v1/push` path from it.

The Vector logging agent used in DKP will automatically add this path when sending data to Loki.

Additional resources

- Module `log-shipper` custom resources
- Module `loki` parameters

Multitenancy

Multitenancy is the ability to create isolated environments (projects) within a Kubernetes cluster. Projects are similar to namespaces but offer more capabilities. While namespaces are used for logical separation of resources in Kubernetes, they do not, for example, restrict network communication, pod resource consumption, or host directory mounts. These limitations make namespaces insufficient for modern development needs. By default, namespaces also do not include logging, auditing, and vulnerability scanning.

Using projects helps address these limitations and offers the following benefits:

- For platform administrators:
  - **Consistency**: Administrators can create projects using a shared template, which ensures consistency and simplifies management.
  - **Security**: Projects ensure isolation for resources and access policies between different tenants, supporting a secure multitenant environment.
  - **Resource consumption**: Administrators can easily set resource quotas and limits for each project to prevent resource overconsumption.
- For platform users:
  - **Immediate start**: Developers can request projects created per templates from administrators to quickly start developing new applications.
  - **Isolation**: Each project provides an isolated environment, allowing developers to deploy and test their applications without affecting others.

> ⚠ Secret copying across all namespaces is incompatible with projects in multitenancy mode.
>
> This mode creates isolated environments for users within their projects, while `secret-copier` automatically distributes secrets to all namespaces. If sensitive data is present in a user's private environment, it could lead to a data leak and a security model breach.

Limitations

Projects has several limitations:

- Creating more than one namespace within a project is not supported. If you need multiple namespaces, create a separate project for each of them.
- Template resources are applied only to a single namespace whose name matches the project name.

Additional resources

- Module `secret-copier` documentation

Project management

Deckhouse Kubernetes Platform includes a set of templates for creating projects:

- `default` is a template for basic project use cases:
  - resource limits
  - network isolation
  - automatic alerts and log collection
  - selecting a security profile
  - configuring project administrators
- `secure` includes all capabilities of the `default` template and adds:
  - configuring allowed UID/GID ranges for the project
  - audit rules for Linux users' interactions with the kernel
  - scanning container images at runtime for known vulnerabilities (CVEs)
- `secure-with-dedicated-nodes` includes all capabilities of the `secure` template and adds:
  - defining a node selector for all pods in the project: when a pod is created, its node selector is automatically **replaced** with the project's node selector
  - defining default tolerations for all pods in the project: when a pod is created, the default tolerations are automatically **added** to it

To list all available parameters for a project template, run:

```
d8 k get projecttemplates <PROJECT_TEMPLATE_NAME> -o
jsonpath='{.spec.parametersSchema.openAPIV3Schema}' | jq
```

Project creation

1. To create a project, create a custom resource Project with the project template name specified in the .spec.projectTemplateName field.
2. In the .spec.parameters parameter, specify the values for the .spec.parametersSchema.openAPIV3Schema section of the custom resource ProjectTemplate.

   An example of creating a project using Project from the `default` ProjectTemplate is shown below:

```
apiVersion: deckhouse.io/v1alpha2
kind: Project
metadata:
  name: my-project
spec:
  description: This is an example from the Deckhouse documentation.
  projectTemplateName: default
  parameters:
    resourceQuota:
      requests:
        cpu: 5
        memory: 5Gi
        storage: 1Gi
      limits:
        cpu: 5
        memory: 5Gi
    networkPolicy: Isolated
    podSecurityProfile: Restricted
    extendedMonitoringEnabled: true
    administrators:
    - subject: Group
      name: k8s-admins
```

3. To check the project status, run the following command:

   ```
   d8 k get projects my-project
   ```

   A successfully created project should have the `Deployed` (synced) status. If the status is `Error`, add the `-o yaml` flag to the command (for example, `d8 k get projects my-project -o yaml`) to get more details about the error cause.

Automatic project creation for a namespace

You can create a new project for an existing namespace. To do this, add the `projects.deckhouse.io/adopt` annotation to the namespace. For example:

1. Create a new namespace:

   ```
   d8 k create ns test
   ```

2. Annotate it with the following command:

   ```
   d8 k annotate ns test projects.deckhouse.io/adopt=""
   ```

3. Make sure the project has been created:

   ```
   d8 k get projects
   ```

   A new project matching the namespace will appear in the list of projects:

   ```
   NAME        STATE      PROJECT TEMPLATE   DESCRIPTION                                          AGE
   deckhouse   Deployed   virtual            This is a virtual project
   181d
   default     Deployed   virtual            This is a virtual project
   181d
   test          Deployed   empty
   1m
   ```

You can change the template of an existing project to another available template.

⚠ Note that changing the template may cause resource conflicts: if the template chart defines resources that already exist in the namespace, the template cannot be applied.

Creating a custom project template

The default project templates cover common baseline scenarios and also serve as examples of what templates can do.

To create your own template:

1. Use one of the default templates as a starting point, for example, `default`.
2. Export it to a separate file, for example, `my-project-template.yaml`, using the following command:

   ```
   d8 k get projecttemplates default -o
   ```

3. Edit the `my-project-template.yaml` file and apply the required changes.

   > **i** You must update not only the template itself, but also the input parameters schema to match it.
   >
   > Project templates support all Helm templating functions.

4. Change the template name in the `.metadata.name` field.
5. Apply the resulting template with the following command:

   ```
   d8 k apply -f my-project-template.yaml
   ```

6. Check that the new template is available by running:

   ```
   d8 k get projecttemplates <NEW_TEMPLATE_NAME>
   ```

Using labels to manage resources

When creating resources in ProjectTemplate, you can use special labels to control how the `multitenancy-manager` processes these resources.

Skipping creation of the `heritage: multitenancy-manager` label

By default, all resources created from ProjectTemplate receive the label `heritage: multitenancy-manager`.
This label prohibits changes to resources by users or any other controller except `multitenancy-manager`.
If you need to allow resource modification (for example, for compatibility with other systems, or if implementing your own control over the created objects), add the label `projects.deckhouse.io/skip-heritage-label` to the resource.

Example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: {{ .projectName }}
  labels:
    projects.deckhouse.io/skip-heritage-label: "true"
    app: my-app
data:
  key: value
```

In this case, the resource will receive the labels `projects.deckhouse.io/project` and `projects.deckhouse.io/project-template`, but will not receive the label `heritage: multitenancy-manager`.

Excluding resources from management by multitenancy-manager

If you need to exclude a resource from management by `multitenancy-manager` (for example, if the resource should be managed manually or by another controller), add the label `projects.deckhouse.io/unmanaged` to the resource.

Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: external-secret
  namespace: {{ .projectName }}
  labels:
    projects.deckhouse.io/unmanaged: "true"
type: Opaque
data:
  token: <base64-encoded-value>
```

Resources with the label `projects.deckhouse.io/unmanaged`:

- Will be created **only once** when the project is created;
- **Will not be updated** with subsequent template changes or updates;
- Will not be monitored in the project's status;
- Will receive the labels `projects.deckhouse.io/project` and `projects.deckhouse.io/project-template` but **will not receive** the label `heritage: multitenancy-manager`.

⚠ Once a resource is marked as `unmanaged`, it will be created on initial installation but not updated when the ProjectTemplate is changed.
After creation, the resource becomes fully independent and must be managed manually.

Implementing validation of object changes with a custom label

The `multitenancy-manager` module uses ValidatingAdmissionPolicy to protect resources labeled `heritage: multitenancy-manager` from manual changes.
You can implement similar validation for resources with any label.

How validation works in multitenancy-manager

Validation occurs for objects labeled `heritage: multitenancy-manager`.
The following components are used for this:

1. ValidatingAdmissionPolicy: Defines validation rules:
   - Operations: `UPDATE` and `DELETE`.
   - Check: only operations on behalf of the controller's service account are allowed.
   - Applies to all resources and API groups.
2. ValidatingAdmissionPolicyBinding: Defines which objects the validation applies to:
   - Uses `namespaceSelector` and `objectSelector` to select resources by the label `heritage: multitenancy-manager`.

Creating your own validation

To implement validation for resources with a different label (for example, `heritage: my-custom-label`):

1. Create a file with the ValidatingAdmissionPolicy and ValidatingAdmissionPolicyBinding resource manifests:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: my-custom-label-validation
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups:   ["*"]
        apiVersions: ["*"]
        operations:  ["UPDATE", "DELETE"]
        resources:   ["*"]
        scope: "*"
  validations:
    - expression: 'request.userInfo.username == "system:serviceaccount:my-
namespace:my-service-account"' # Replace with your service account
      reason: Forbidden
      messageExpression: 'object.kind == ''Namespace'' ? ''This resource is
managed by '' + object.metadata.name + '' system. Manual modification is
forbidden.''
        : ''This resource is managed by '' + object.metadata.namespace + ''
system. Manual modification is forbidden.'''
---
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
  name: my-custom-label-validation
spec:
  policyName: my-custom-label-validation
  validationActions: [Deny, Audit]
  matchResources:
    namespaceSelector:
      matchLabels:
        heritage: my-custom-label
    objectSelector:
      matchLabels:
        heritage: my-custom-label
```

2. Configure the validation parameters:
   - `policyName`: Unique policy name (must match in Policy and Binding).
   - `request.userInfo.username`: The name of the service account allowed to change resources (replace with your service account).
   - `heritage: my-custom-label`: The value of the `heritage` label for your resources (replace with your value). The use of the values `multitenancy-manager`, `deckhouse` is prohibited.
   - `failurePolicy: Fail`: Policy on validation failure.
     - `Fail`: Reject the request on validation failure.
     - `Ignore`: Ignore validation errors.
   - `validationActions`: Validation actions:
     - `Deny`: Deny unauthorized operations.
     - `Audit`: Record operations in the audit log.
3. Apply the policy:

```
d8 k apply -f my-validation-policy.yaml
```

4. Ensure your resources have the corresponding `heritage` label:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-resource
  labels:
    heritage: my-custom-label
```

Granting cluster-scoped resources to projects

The `multitenancy-manager` lets cluster administrators control which cluster-scoped resources (for example StorageClass) may be referenced from within project namespaces.

To do this, custom resources are used:

- GrantableClusterResourceDefinition (cluster-scoped) — registers a cluster resource that can be granted to projects: which resource it is (`grantedResource`), where references to it are validated (`usageReferences`), the baseline availability (`defaultAvailability`), and how the per-project default is discovered (`defaultFrom`). Each reference opts into defaulting individually with `default: true` — set it only for a field whose value the resource always needs (such as a `PersistentVolumeClaim`'s `storageClassName`). Leave it off for a reference whose absence is meaningful, such as an annotation that merely toggles a feature; that reference is still validated and counted, just never filled in.
- ClusterResourceGrantPolicy (cluster-scoped) — selects projects (by namespace labels via `projectSelector`) and, per resource (`resourceName`), the granted names (`allowed`, `allowedSelector`) and the per-project `default`. An allow-list restricts the resource to it.
- AvailableClusterResource (namespaced, read-only, short name `available`) — the controller-rendered catalog of what a project may use; tenants read it to discover the available names.
- ClusterResourceGrant (namespaced) — the per-project object-quota pool (limits on object count and on measured quantities such as requested storage); its status reports current usage.

```
apiVersion: multitenancy.deckhouse.io/v1alpha1
kind: GrantableClusterResourceDefinition
metadata:
  name: storageclasses
spec:
  grantedResource:
    apiGroup: storage.k8s.io
    kind: StorageClass
  enforcement: Managed
  defaultAvailability: All
  defaultFrom:
    annotationKey: storageclass.kubernetes.io/is-default-class
  usageReferences:
    - rule:
        apiGroups:
          - ""
        apiVersions:
          - v1
        resources:
          - persistentvolumeclaims
      fieldPath: $.spec.storageClassName
      default: true
---
apiVersion: multitenancy.deckhouse.io/v1alpha1
kind: ClusterResourceGrantPolicy
metadata:
  name: production-storage
spec:
  projectSelector:
    matchLabels:
      environment: production
  resources:
    - resourceName: storageclasses
      default: fast-ssd        # Overrides the annotation-based default.
      allowed:
        - fast-ssd
        - standard
      allowedSelector:         # Plus any StorageClass with label shared=true.
        matchLabels:
          shared: "true"
```

Enforcement notes:

- The validating webhook denies creating/updating objects in matched projects whose referenced value is not granted. On update, values already present in the object are grandfathered in, so pre-existing objects are not broken.
- The defaulting webhook fills in the granted default on creation only, and only into references marked `default: true`. References left without it (such as feature-toggling annotations) are never filled in.
- A grant that matches no project, or a project with no matching grant, imposes no restriction.

Additional resources

- Module `multitenancy-manager` custom resources

Backup and restore

Manual cluster restore

Restoring a cluster with a single control plane node

To properly restore the cluster, follow these steps on the master node:

1. Prepare the `etcdutl` utility. Locate and copy the executable on the node:

```
cp $(find /var/lib/containerd/ -name etcdutl -print -quit) /usr/local/bin/etcdutl
```

Check the version of `etcdutl`:

```
etcdutl version
```

Make sure the output of `etcdutl version` is displayed without errors.

If `etcdutl` is not found, download the binary from the official etcd repository, choosing a version that matches your cluster's etcd version:

```
wget "https://github.com/etcd-io/etcd/releases/download/v3.6.1/etcd-v3.6.1-linux-amd64.tar.gz"
tar -xzvf etcd-v3.6.1-linux-amd64.tar.gz && mv etcd-v3.6.1-linux-amd64/etcdutl /usr/local/bin/etcdutl
```

2. Check the etcd version in the cluster (if the Kubernetes API is accessible):

```
d8 k -n kube-system exec -ti etcd-$(hostname) -- etcdutl version
```

If the command executes successfully, it will display the current etcd version.

3. Stop etcd. Move the etcd manifest to prevent kubelet from launching the etcd pod:

```
mv /etc/kubernetes/manifests/etcd.yaml ~/etcd.yaml
```

4. Make sure the etcd pod is stopped:

```
crictl ps | grep etcd
```

If the command returns nothing, the etcd pod has been successfully stopped.

5. Backup current etcd data. Create a backup copy of the `member` directory:

```
cp -r /var/lib/etcd/member/ /var/lib/deckhouse-etcd-backup
```

This backup will allow you to roll back in case of issues.

6. Place the etcd snapshot file. Copy or move the `etcd-backup.snapshot` file to the current user's (root) home directory:

```
cp /path/to/backup/etcd-backup.snapshot ~/etcd-backup.snapshot
```

Ensure the file is readable:

```
ls -la ~/etcd-backup.snapshot
```

7. Clean the etcd directory. Remove old data to prepare for restore:

```
rm -rf /var/lib/etcd
```

Verify that `/var/lib/etcd` is now empty or does not exist:

```
ls -la /var/lib/etcd
```

8. Restore the etcd database from the snapshot using `etcdutl`:

```
ETCDCTL_API=3 etcdutl snapshot restore ~/etcd-backup.snapshot  --data-dir=/var/lib/etcd
```

After the command completes, check that files have appeared in `/var/lib/etcd/`, reflecting the restored state.

9. Start etcd. Move the manifest back so that kubelet relaunches the etcd pod:

   ```
   mv ~/etcd.yaml /etc/kubernetes/manifests/etcd.yaml
   ```

10. Wait for the pod to be created and reach `Running` state. Make sure it is up and running:

    ```
    crictl ps --label io.kubernetes.pod.name=etcd-$HOSTNAME
    ```

    Pod startup may take some time. Once etcd is running, the cluster will be restored from the snapshot.

    Output example:

    ```
    CONTAINER        IMAGE            CREATED              STATE     NAME
    ATTEMPT     POD ID          POD
    4b11d6ea0338f    16d0a07aa1e26    About a minute ago   Running   etcd      0
    ee3c8c7d7bba6   etcd-gs-test
    ```

11. Restart the master node.

Restoring a multi-master cluster

To properly restore a multi-master cluster, follow these steps:

1. Enable High Availability (HA) mode. This is necessary to preserve at least one Prometheus replica and its PVC, since HA is disabled by default in single-master clusters.
2. Switch the cluster to single master mode:
   - In a cloud cluster, follow the instructions.
   - In a static cluster, remove any unnecessary master nodes from the `control-plane` role by following the instructions and then remove them from the cluster.
   - In a static cluster with the configured HA mode based on two master nodes and an arbiter node, remove the arbiter node and additional master nodes.
   - In a cloud cluster with the configured HA mode based on two master nodes and an arbiter node, use the instructions to remove the additional master nodes and the arbiter node.
3. Restore etcd from the backup on the only remaining master node. Follow the instructions for restoring a cluster with a single control-plane node.
4. Once etcd is restored, remove the records of the previously deleted master nodes from the cluster using the following command (replace with the actual node name):

   ```
   d8 k delete node <MASTER_NODE_NAME>
   ```

   > **Warning.** If the `d8 k` or `kubectl` commands are unavailable on the node, check the `/etc/kubernetes/kubernetes-api-proxy/upstreams.json` configuration file. It should only specify your current API server. If the configuration contains lines with IP addresses of old master nodes, remove them. Edit the configuration in a similar way on all other nodes.

5. Reboot the master node. Ensure that the other nodes transition to the `Ready` state.
6. Wait for Deckhouse to process all tasks in the queue:

   ```
   d8 system queue main
   ```

7. Switch the cluster back to multi-master mode. For cloud clusters, follow the instructions.

Once you go through these steps, the cluster will be successfully restored in the multi-master configuration.

Restoring individual objects

Restoring Kubernetes objects from an etcd backup

To restore individual cluster objects (e.g., specific Deployments, Secrets, or ConfigMaps) from an etcd

snapshot, follow these steps:

1. Launch a temporary etcd instance. Create a separate etcd instance that runs independently from the main cluster.
2. Load data from backup copy into temporary etcd instance. Use the existing etcd snapshot file to populate the temporary instance with the necessary data.
3. Unload the manifests of the necessary objects in YAML format.
4. Restore cluster objects from uploaded YAML files.

Example of steps to restore objects from an etcd backup

In the example below, `etcd-backup.snapshot` is a etcd shapshot, `infra-production` is the namespace in which objects need to be restored.

- To decode objects from `etcd` you would need auger. It can be built from source on any machine that has Docker installed (it cannot be done on cluster nodes).

  ```
  git clone -b v1.0.1 --depth 1 https://github.com/etcd-io/auger
  cd auger
  make release
  build/auger -h
  ```

- Resulting executable `build/auger`, and also the `snapshot` from the backup copy of etcd must be uploaded on master-node, on which following actions would be performed.

Following actions are performed on a master node, to which `etcd snapshot` file and `auger` tool were copied:

1. Set the correct access permissions for the backup file:

   ```
   chmod 644 etcd-backup.snapshot
   ```

2. Set full path for snapshot file and for the tool into environmental variables:

   ```
   SNAPSHOT=/root/etcd-restore/etcd-backup.snapshot
   AUGER_BIN=/root/auger
   chmod +x $AUGER_BIN
   ```

3. Run a Pod with temporary instance of `etcd`.
   - Create Pod manifest. It should schedule on current master node by `$HOSTNAME` variable, and mounts snapshot file by `$SNAPSHOT` variable, which it then restores in temporary `etcd` instance:

```
cat <<EOF >etcd.pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: etcdrestore
  namespace: default
spec:
  nodeName: $HOSTNAME
  tolerations:
  - operator: Exists
  initContainers:
  - command:
    - etcdutl
    - snapshot
    - restore
    - "/tmp/etcd-snapshot"
    - --data-dir=/default.etcd
    image: $(kubectl -n kube-system get pod -l component=etcd -o jsonpath="
{.items[*].spec.containers[*].image}" | cut -f 1 -d ' ')
    imagePullPolicy: IfNotPresent
    name: etcd-snapshot-restore
    # Uncomment the fragment below to set limits for the container if the
node does not have enough resources to run it.
    # resources:
    #   requests:
    #     ephemeral-storage: "200Mi"
    #   limits:
    #     ephemeral-storage: "500Mi"
    volumeMounts:
    - name: etcddir
      mountPath: /default.etcd
    - name: etcd-snapshot
      mountPath: /tmp/etcd-snapshot
      readOnly: true
  containers:
  - command:
    - etcd
    image: $(kubectl -n kube-system get pod -l component=etcd -o jsonpath="
{.items[*].spec.containers[*].image}" | cut -f 1 -d ' ')
    imagePullPolicy: IfNotPresent
    name: etcd-temp
    volumeMounts:
    - name: etcddir
      mountPath: /default.etcd
  volumes:
  - name: etcddir
    emptyDir: {}
    # Use the snippet below instead of emptyDir: {} to set limits for the
container if the node's resources are insufficient to run it.
    # emptyDir:
    #  sizeLimit: 500Mi
  - name: etcd-snapshot
    hostPath:
      path: $SNAPSHOT
      type: File
EOF
```

- Create Pod from the resulting manifest:

```
d8 k create -f etcd.pod.yaml
```

4. Set environment variables. In this example:
   - `infra-production` — namespace which we will search resources in.
   - `/root/etcd-restore/output` — path for outputting recovered resource manifests.
   - `/root/auger` — path to `auger` executable.

   ```
   FILTER=infra-production
   BACKUP_OUTPUT_DIR=/root/etcd-restore/output
   mkdir -p $BACKUP_OUTPUT_DIR && cd $BACKUP_OUTPUT_DIR
   ```

5. Commands below will filter needed resources by `$FILTER` and output them into `$BACKUP_OUTPUT_DIR` directory:

   ```
   files=($(kubectl -n default exec etcdrestore -c etcd-temp -- etcdctl  --
   endpoints=localhost:2379 get / --prefix --keys-only | grep "$FILTER"))
   for file in "${files[@]}"
   do
     OBJECT=$(kubectl -n default exec etcdrestore -c etcd-temp -- etcdctl  --
   endpoints=localhost:2379 get "$file" --print-value-only | $AUGER_BIN decode)
     FILENAME=$(echo $file | sed -e "s#/registry/##g;s#/#_#g")
     echo "$OBJECT" > "$BACKUP_OUTPUT_DIR/$FILENAME.yaml"
     echo $BACKUP_OUTPUT_DIR/$FILENAME.yaml
   done
   ```

6. Restore objects from exported YAML files.
7. Delete the Pod with a temporary instance of etcd:

   ```
   d8 k -n default delete pod etcdrestore
   ```

Restoring cluster objects from exported YAML files

To restore objects from exported YAML files, follow these steps:

1. Prepare the YAML files for restoration. Before applying them to the cluster, remove technical fields that may be outdated or interfere with the recovery process:
   - `creationTimestamp`
   - `UID`
   - `status`

   You can edit these fields manually or use YAML/JSON processing tools such as `yq` or `jq`.
2. Create the objects in the cluster. To restore individual resources, run:

   ```
   d8 k create -f <PATH_TO_FILE>.json
   ```

   You can specify either a single file or a directory path.
3. To restore multiple objects at once, use the `find` command:

   ```
   find $BACKUP_OUTPUT_DIR -type f -name "*.yaml" -exec d8 k create -f {} \;
   ```

   This will locate all `.yaml` files within the specified `$BACKUP_OUTPUT_DIR` and apply them sequentially using `d8 k create`.

After completing these steps, the selected objects will be recreated in the cluster based on the definitions in the YAML files.

Restoring objects after changing the master node IP address

> ⚠ This section describes a scenario where only the IP address of the master node has changed, and all other objects in the etcd backup (such as CA certificates) remain valid. It assumes the restoration is performed in a single-master-node cluster.

To restore etcd objects after changing the master node's IP address, follow these steps:

1. Restore etcd from the backup. Use the standard etcd restore procedure with a snapshot. Make sure not to change any parameters during restoration other than the etcd data itself.
2. Update the IP address in static configuration files:
   - Check the Kubernetes component manifest files located in `/etc/kubernetes/manifests/`.
   - Review kubelet's system configuration files (typically found in `/etc/systemd/system/kubelet.service.d/` or similar directories).
   - Update the IP address in any other configurations that reference the old address, if necessary.
3. Regenerate certificates that were issued for the old IP. Delete or move old certificates related to the API server and etcd (if applicable). Then generate new certificates, specifying the new master node IP address as a SAN (Subject Alternative Name).
4. Restart all services that use the updated configurations and certificates. Force kubelet to restart control-plane manifests (API server, etcd, etc.). Either restart the system services manually (e.g., `systemctl restart kubelet`) or ensure they restart automatically.
5. Wait for kubelet to regenerate its own certificate.

These actions can be performed either automatically using a script, or manually by running the required commands step-by-step.

Automated object extraction when changing IP address

To simplify cluster recovery after the master node's IP address changes, use the script provided below. Before running the script:

1. Specify the correct paths and IP addresses:
   - `ETCD_SNAPSHOT_PATH`: The path to the etcd snapshot backup.
   - `OLD_IP`: The old master node IP address used when the backup was created.
   - `NEW_IP`: The new IP address of the master node.
2. Make sure the Kubernetes version (`KUBERNETES_VERSION`) matches the one used in the cluster. This is necessary for downloading the correct version of kubeadm.
3. Download `etcdutl` if it is not installed.
4. After running the script, wait for the kubelet to regenerate its certificate with the new IP address. You can verify this in the `/var/lib/kubelet/pki/` directory, where a new certificate should appear.

Object extraction script

```
ETCD_SNAPSHOT_PATH="./etcd-backup.snapshot" # Path to the etcd snapshot.
OLD_IP=10.242.32.34                    # Old master node IP address.
NEW_IP=10.242.32.21                    # New master node IP address.
KUBERNETES_VERSION=1.28.0                # Kubernetes version.

mv /etc/kubernetes/manifests/etcd.yaml ~/etcd.yaml
mkdir ./etcd_old
mv /var/lib/etcd ~/etcd_old
ETCDUTL_PATH=$(find /var/lib/containerd/ -name etcdutl -print -quit)

ETCDCTL_API=3 $ETCDUTL_PATH snapshot restore etcd-backup.snapshot --data-dir=/var/lib/etcd

mv ~/etcd.yaml /etc/kubernetes/manifests/etcd.yaml

find /etc/kubernetes/ -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
find /etc/systemd/system/kubelet.service.d -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
find  /var/lib/bashible/ -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'

mkdir -p ./old_certs/etcd
mv /etc/kubernetes/pki/apiserver.* ./old_certs/
mv /etc/kubernetes/pki/etcd/server.* ./old_certs/etcd/
mv /etc/kubernetes/pki/etcd/peer.* ./old_certs/etcd/

curl -LO https://dl.k8s.io/v$KUBERNETES_VERSION/bin/linux/amd64/kubeadm
chmod +x kubeadm
./kubeadm init phase certs all --config /etc/kubernetes/deckhouse/kubeadm/config.yaml

crictl ps --name 'kube-apiserver' -o json | jq -r '.containers[0].id' | xargs crictl stop
crictl ps --name 'kubernetes-api-proxy' -o json | jq -r '.containers[0].id' | xargs crictl
stop
crictl ps --name 'etcd' -o json | jq -r '.containers[].id' | xargs crictl stop

systemctl daemon-reload
systemctl restart kubelet.service
```

Manual object restore after changing the IP address

If you prefer to manually make changes during cluster recovery after the master node’s IP address has changed, follow these steps:

1. Restore etcd from the backup:
   - Move the etcd manifest so that kubelet stops the corresponding pod:

     ```
     mv /etc/kubernetes/manifests/etcd.yaml ~/etcd.yaml
     ```

   - Create a directory to temporarily store the previous etcd data:

     ```
     mkdir ./etcd_old
     mv /var/lib/etcd ./etcd_old
     ```

   - Find or download the `etcdutl` utility if it’s not available, and perform the snapshot restore:

     ```
     ETCD_SNAPSHOT_PATH="./etcd-backup.snapshot" # Path to the etcd snapshot.
     ETCDUTL_PATH=$(find /var/lib/containerd/ -name etcdutl -print -quit)

     ETCDCTL_API=3 $ETCDUTL_PATH snapshot restore \
       etcd-backup.snapshot \
       --data-dir=/var/lib/etcd
     ```

   - Restore the etcd manifest so kubelet starts the etcd pod again:

     ```
     mv ~/etcd.yaml /etc/kubernetes/manifests/etcd.yaml
     ```

- Verify etcd is running by checking the pod list using `crictl ps | grep etcd` or reviewing the kubelet logs.

2. Update the IP address in static configuration files. If the old IP address is used in manifests or kubelet services, replace it with the new one:

```
OLD_IP=10.242.32.34                     # Old master node IP address.
NEW_IP=10.242.32.21                     # New master node IP address.

find /etc/kubernetes/ -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
find /etc/systemd/system/kubelet.service.d -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {}
';'
find  /var/lib/bashible/ -type f -exec sed -i "s/$OLD_IP/$NEW_IP/g" {} ';'
```

3. Regenerate certificates issued for the old IP address:

   - Prepare a directory to store the old certificates:

     ```
     mkdir -p ./old_certs/etcd
     mv /etc/kubernetes/pki/apiserver.* ./old_certs/
     mv /etc/kubernetes/pki/etcd/server.* ./old_certs/etcd/
     mv /etc/kubernetes/pki/etcd/peer.* ./old_certs/etcd/
     ```

   - Install or download kubeadm to match the current Kubernetes version:

     ```
     KUBERNETES_VERSION=1.28.0 # Kubernetes version.
     curl -LO https://dl.k8s.io/v$KUBERNETES_VERSION/bin/linux/amd64/kubeadm
     chmod +x kubeadm
     ```

   - Generate new certificates with the updated IP:

     ```
     ./kubeadm init phase certs all --config /etc/kubernetes/deckhouse/kubeadm/config.yaml
     ```

     The new IP address will be included in the generated certificates.

4. Restart all services that use the updated configuration and certificates. To immediately stop active containers, run:

```
 crictl ps --name 'kube-apiserver' -o json | jq -r '.containers[0].id' | xargs crictl
stop
 crictl ps --name 'kubernetes-api-proxy' -o json | jq -r '.containers[0].id' | xargs
crictl stop
 crictl ps --name 'etcd' -o json | jq -r '.containers[].id' | xargs crictl stop

 systemctl daemon-reload
 systemctl restart kubelet.service
```

   Kubelet will restart the necessary pods, and Kubernetes components will load the new certificates.

5. Wait for kubelet to regenerate its own certificate. Kubelet will automatically generate a new certificate with the updated IP address:

   - Check the `/var/lib/kubelet/pki/` directory.
   - Ensure the new certificate is present and valid.

Once these steps are completed, the cluster will be successfully restored and fully functional with the new master node IP address.

Creating backups with Deckhouse CLI

Deckhouse CLI (`d8`) provides the `backup` command for creating backups of various cluster components:

- `etcd`: Snapshot of the Deckhouse key-value data store.
- `cluster-config`: Archive containing key configuration objects of the cluster.
- `loki`: Export of logs from the built-in Loki API.

Backing up etcd

An etcd snapshot allows you to preserve the current state of the cluster at the key-value storage level. This is a full dump that can be used for recovery.

To create a snapshot, run the following command:

```
d8 backup etcd <path-to-snapshot> [flags]
```

Flags:

- `-p`, `--etcd-pod string`: Name of the etcd pod to snapshot.
- `-h`, `--help`: Show help for the etcd command.
- `--verbose`: Enable verbose output for detailed logging.

Example:

```
d8 backup etcd etcd-backup.snapshot
```

Example output:

```
2025/04/22 08:38:58 Trying to snapshot etcd-sandbox-master-0
2025/04/22 08:39:01 Snapshot successfully taken from etcd-sandbox-master-0
```

Automatic etcd backup

Deckhouse automatically performs a daily etcd backup using a CronJob that runs inside the `d8-etcd-backup` pod in the `kube-system` namespace. The job creates a snapshot of the database, compresses it, and saves the archive locally on the node at `/var/lib/etcd/`:

```
etcdctl snapshot save etcd-backup.snapshot
tar -czvf etcd-backup.tar.gz etcd-backup.snapshot
mv etcd-backup.tar.gz /var/lib/etcd/etcd-backup.tar.gz
```

To configure automatic etcd backups, use the `control-plane-manager` module. The required parameters are specified in its configuration:

Parameter	Description
`etcd.backup.enabled`	Enables daily etcd backup.
`etcd.backup.cronSchedule`	Cron-formatted schedule for running the backup. Local time of `kube-controller-manager` is used.
`etcd.backup.hostPath`	Path on master nodes where etcd backup archives will be stored.

Example configuration fragment:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    etcd:
      backup:
        cronSchedule: 0 14 * * *
        enabled: true
```

Cluster configuration backup

The `d8 backup cluster-config` command creates an archive containing a set of key resources related to the cluster configuration. This is not a full backup of all objects, but a specific whitelist.

To create the backup, run the following command:

```
d8 backup cluster-config <path-to-backup-file>
```

Example:

```
d8 backup cluster-config /backup/cluster-config-2025-04-21.tar
```

The archive includes only those objects that meet the following criteria:

- CustomResource objects whose CRDs are annotated with:

  ```
  backup.deckhouse.io/cluster-config=true
  ```

- StorageClasses with the label:

  ```
  heritage=deckhouse
  ```

- Secrets and ConfigMaps from namespaces starting with `d8-` or `kube-`, if they are explicitly listed in the whitelist file.
- Cluster-level Roles and Bindings (ClusterRole and ClusterRoleBinding), if they are not labeled with:

  ```
  heritage=deckhouse
  ```

i The backup includes only CR objects, but not the CRD definitions themselves. To fully restore the cluster, the corresponding CRDs must already be present (e.g., installed by Deckhouse modules).

Example whitelist content:

Namespace	Object	Name
`d8-system`	Secret	`d8-cluster-terraform-state`
		`$regexp:^d8-node-terraform-state-(.*)$`
		`deckhouse-registry`
	ConfigMap	`d8-deckhouse-version-info`
`kube-system`	ConfigMap	`d8-cluster-is-bootstraped`
		`d8-cluster-uuid`
		`extension-apiserver-authentication`
	Secret	`d8-cloud-provider-discovery-data`
		`d8-cluster-configuration`
		`d8-cni-configuration`
		`d8-control-plane-manager-config`
		`d8-node-manager-cloud-provider`
		`d8-pki`
		`d8-provider-cluster-configuration`

Namespace	Object	Name
		`d8-static-cluster-configuration`
		`d8-secret-encryption-key`
`d8-cert-manager`	Secret	`cert-manager-letsencrypt-private-key`
		`selfsigned-ca-key-pair`

Exporting logs from Loki

The `d8 backup loki` command is intended for exporting logs from the built-in Loki. This is not a full backup, but rather a diagnostic export: the resulting data cannot be restored back into Loki.

To perform the export, `d8` accesses the Loki API using the `loki` ServiceAccount in the `d8-monitoring` namespace, authenticated via a token stored in a Kubernetes secret.

The `loki` ServiceAccount is automatically created starting from Deckhouse v1.69.0. However, to use the `d8 backup loki` command, you must manually create the token secret and assign the necessary Role and RoleBinding if they are not already present.

Apply the manifests below before running `d8 backup loki` to ensure the command can properly authenticate and access the Loki API.

Example manifests:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: loki-api-token
  namespace: d8-monitoring
  annotations:
    kubernetes.io/service-account.name: loki
type: kubernetes.io/service-account-token
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: access-to-loki-from-d8
  namespace: d8-monitoring
rules:
  - apiGroups: ["apps"]
    resources:
      - "statefulsets/http"
    resourceNames: ["loki"]
    verbs: ["create", "get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: access-to-loki-from-d8
  namespace: d8-monitoring
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: access-to-loki-from-d8
subjects:
  - kind: ServiceAccount
    name: loki
    namespace: d8-monitoring
```

To create a log backup, run the following command:

```
d8 backup loki [flags]
```

Example:

```
d8 backup loki --days 1 > ./loki.log
```

Flags:

- `--start`, `--end`: Time range boundaries in the format “YYYY-MM-DD HH:MM:SS”.
- `--days`: The time window size for log export (default is 5 days).
- `--limit`: The maximum number of log lines per request (default is 5000).

You can list all available flags using the following command:

```
d8 backup loki --help
```

Additional resources

- Module `control-plane-manager` documentation

Automatic assignment of namespace labels and annotations

You can automate the assignment of labels and annotations to namespaces in a Deckhouse cluster based on predefined patterns. This can be useful, for example, to automatically include new namespaces in monitoring without having to manually edit each one.

How it works

- All namespaces whose names match the patterns in `includeNames` and do not match patterns in `excludeNames` automatically receive the specified labels and annotations.
- When the configuration is changed, labels and annotations of existing namespaces are updated accordingly.
- Newly created namespaces that match the defined patterns also receive the specified labels and annotations automatically.

Configuring automatic label and annotation assignment

Enable the `namespace-configurator` module:

```
d8 system module enable namespace-configurator
```

Configure automatic label and annotation assignment in ModuleConfig `namespace-configurator`:

1. Specify the annotations and labels to be applied to namespaces in the fields `settings.configurations.annotations` and `settings.configurations.labels`, respectively;
2. Define the matching rules for namespace names:
   - In `includeNames`, list regular expressions for names you want to match.
   - In `excludeNames`, list names that should be excluded.

Example configuration

In the following configuration example, automatic addition of the label `extended-monitoring.deckhouse.io/enabled=true` and the annotation `foo=bar` is configured for all namespaces whose names start with `prod-` or `infra-`, except for `infra-test`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: namespace-configurator
spec:
  version: 1
  enabled: true
  settings:
    configurations:
    - annotations:
        foo: bar
      labels:
        extended-monitoring.deckhouse.io/enabled: "true"
      includeNames:
      - "^prod"
      - "^infra"
      excludeNames:
      - "infra-test"
```

Additional resources

- Module `namespace-configurator` documentation

Application delivery with Argo CD

This section describes application delivery with Argo CD in Deckhouse Kubernetes Platform (DKP).

Argo CD is an Open Source tool for continuous delivery of applications to Kubernetes that implements the GitOps approach. A Git repository is the source of truth for describing applications, their configuration, and target environments.

Argo CD tracks changes, shows differences between the desired and actual resource state, and when needed brings the cluster to the target state automatically or manually.

Main Argo CD capabilities:

- automatic and manual application deployment in Kubernetes;
- visualization of differences between the desired and actual state;
- tracking application configuration in Git;
- working with multiple Kubernetes clusters;
- access control through its own role model;
- rollback to a previously recorded configuration and action audit;
- working through the web interface, CLI, and API, including integration with Git webhooks and external automation systems.

In DKP, Argo CD is run with the operator-argo module, based on the Argo CD Operator project. The module lets you declaratively deploy and maintain one or more Argo CD instances in a DKP cluster without manually installing and maintaining components.

Additionally, the module provides:

- single sign-on (SSO) support;
- integration with monitoring in DKP;
- high availability mode;
- support for automatic image updates via Argo CD Image Updater;
- access control for cluster resources.

A typical administrator setup scenario:

1. Enable the `operator-argo` module and create an ArgoCD object.
2. Configure user authentication.
3. Define the role-based access model.
4. Give users access to the Argo CD instance for application delivery (for more details, see the Usage section).

The design of the `operator-argo` module is described in the Architecture section. Module parameters and custom resources are documented in the `operator-argo` module documentation.

Additional resources

- Official Argo CD website
- Official Argo CD Operator website
- Module `operator-argo` documentation

Running Argo CD

This section describes the steps required to run an Argo CD instance in a DKP cluster:

- Preparing to run Argo CD
- Deploying one or multiple Argo CD instances
- Additional settings for Argo CD

Preparing to run Argo CD

Before creating Argo CD instances, complete the following steps:

1. Enable the `operator-argo` module.
2. Wait until the module switches to the `Ready` state.

   You can check the module state in the DKP web interface or with the following command:

   ```
   d8 k get module operator-argo -W
   ```

Detailed information about module settings is available in the `operator-argo` module documentation.

After you enable the `operator-argo` module, Argo CD custom resources become available in the DKP cluster.

To run an Argo CD instance, create an ArgoCD object. Working with custom resources that belong to an Argo CD instance is described in the Usage section.

List of Argo CD custom resources created by the `operator-argo` module...

CRD	Scope	Purpose
`argocds.argoproj.io`	**`operator-argo` module**	A custom resource used by the `operator-argo` module to configure and maintain an Argo CD instance in the target environment. Installation parameters, component set, integrations, access policies, and other instance settings are defined through the ArgoCD object.
`appprojects.argoproj.io`	**Argo CD instance**	Used for logical segmentation of applications and for defining policies: allowed Git repositories, target clusters and namespaces, as well as access rules and restrictions on resource usage.
`applications.argoproj.io`	**Argo CD instance**	The main Argo CD application CRD for describing an application that must be synchronized from a declarative source (Git, Helm, Kustomize, and others) into Kubernetes. Defines the source, target environment, and synchronization parameters.
`applicationsets.argoproj.io`	**Argo CD instance**	Used for automated creation of a set of Application objects from a template. Suitable for mass application management scenarios across multiple clusters, environments, directories, teams, or repository branches.
`argocdexports.argoproj.io`	**Argo CD instance**	Used for declarative export of data related to an Argo CD instance to external systems or related DKP components. Typically applied in integration scenarios where configuration, status, or access information must be published in a formalized way.

CRD	Scope	Purpose
`namespacemanagements.argoproj.io`	**Argo CD instance**	Used to manage the lifecycle of namespaces within GitOps processes. Can automate creation, configuration, and maintenance of namespaces into which applications are then deployed.
`notificationsconfigurations.argoproj.io`	**Argo CD instance**	Used to configure the Argo CD notification mechanism. Lets you declaratively describe delivery channels and rules for sending events related to synchronization, deployment errors, application status changes, and other operational events.
`imageupdaters.argocd-image-updater.argoproj.io`	**Argo CD instance**	Used to automatically track new container image versions and update application parameters according to the defined versioning and publishing strategy.

Deploying an Argo CD instance

Parameters available for configuring an Argo CD instance are listed in the `operator-argo` module documentation.

To deploy an Argo CD instance in the `argocd` namespace and publish the Argo CD web interface through Ingress, use the following example (provide your own values for `<ARGOCD_DOMAIN>` and `<TLS_SECRET_NAME>`):

```
apiVersion: v1
kind: Namespace
metadata:
  name: argocd
---
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  server:
    host: <ARGOCD_DOMAIN>
    ingress:
      enabled: true
      tls:
        - hosts:
            - <ARGOCD_DOMAIN>
          secretName: <TLS_SECRET_NAME>
    insecure: true
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: argocd-ingress
  namespace: argocd
spec:
  dnsNames:
    - <ARGOCD_DOMAIN>
  issuerRef:
    kind: ClusterIssuer
    name: letsencrypt
  secretName: <TLS_SECRET_NAME>
```

⚠ The `spec.server.insecure: true` parameter in the example above disables internal TLS on the Argo CD API server. This helps avoid redirect loops when publishing through Ingress.

After you create the ArgoCD object, Argo CD components are started in the `argocd` namespace:

```
d8 k -n argocd get pods
NAME                                  READY   STATUS    RESTARTS   AGE
argocd-application-controller-0       1/1     Running   0          35m
argocd-dex-server-759fff8444-zglp4    1/1     Running   0          2d23h
argocd-redis-568f5b889c-jg5dr         1/1     Running   0          4d
argocd-repo-server-78d9d6bcc6-9rwcm   1/1     Running   0          3d22h
argocd-server-76597597f9-kfqdl        1/1     Running   0          35m
podinfo-ccdb96645-zv5tm               1/1     Running   0          2d20h
```

When all components switch to the `Running` status, the web interface of the Argo CD instance becomes available at the address specified in the `spec.server.ingress.tls.hosts` parameter (in the example — `https://<ARGOCD_DOMAIN>`). Authentication setup and credentials retrieval are described in the Configuring authentication and authorization section.

Application delivery with Argo CD is described in the Usage section.

> **i** You can work with Argo CD not only through the web interface and custom resources, but also with the `argocd` CLI utility. You can download the binary from the “Documentation” section of the Argo CD web interface. To get help for the CLI utility, run `argocd --help`.

Deploying multiple Argo CD instances

If the cluster needs multiple Argo CD instances, create a separate namespace (or DKP project) and a separate ArgoCD object for each of them.

For example, create:

- a separate instance for the production environment;
- a separate instance for test environments;
- a separate instance for a specific team or project.

With this approach, each Argo CD instance is managed independently and has its own configuration.

> ⚠ Creating more than one ArgoCD object in a single namespace is not supported.

Advanced settings

Enabling high availability mode

To enable high availability mode, set the `spec.ha.enabled: true` parameter in the ArgoCD object.

Example:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  ha:
    enabled: true
    resources:
      requests:
        cpu: "500m"
        memory: "512Mi"
      limits:
        cpu: "1"
        memory: "1Gi"
```

> ⚠ High availability mode provides high availability only for the `Redis` state store using `HAProxy`. It does not automatically make all Argo CD components highly available.

> **i** Running Argo CD in high availability mode requires at least three cluster nodes because of `pod anti-affinity` rules. Clusters that use IPv6 only are not supported.
>
> When high availability mode is enabled, changes in `.spec.redis.resources` are not applied. Configure resource limits and requests for `Redis` through the `.spec.ha.resources` parameter.

Granting access to cluster resources

By default, an Argo CD instance receives privileges only for the namespace where it runs and for namespaces labeled with `argocd.argoproj.io/managed-by`. The label value must match the name of the namespace where the Argo CD instance runs.

To allow creation of cluster-wide resources, specify the namespace of the target Argo CD instance in the `clusterConfigNamespaces` parameter of the `operator-argo` module settings.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: operator-argo
spec:
  enabled: true
  settings:
    clusterConfigNamespaces: argocd
  version: 1
```

If there are multiple Argo CD instances, list them in the `clusterConfigNamespaces` parameter as a comma-separated list.

After you change the `ModuleConfig`, the following ClusterRole and ClusterRoleBinding objects are created automatically:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    argocds.argoproj.io/name: argocd
    argocds.argoproj.io/namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
    app.kubernetes.io/name: argocd
    app.kubernetes.io/part-of: argocd
  name: argocd-argocd-argocd-application-controller
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- apiGroups:
  - ""
  resources:
  - serviceaccounts
  verbs:
  - impersonate
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    argocds.argoproj.io/name: argocd
    argocds.argoproj.io/namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
```

```
    app.kubernetes.io/name: argocd-application-controller
    app.kubernetes.io/part-of: argocd
  name: argocd-argocd-argocd-application-controller

roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: argocd-argocd-argocd-application-controller
subjects:
- kind: ServiceAccount
  name: argocd-argocd-application-controller
  namespace: argocd
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    argocds.argoproj.io/name: argocd
    argocds.argoproj.io/namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
    app.kubernetes.io/name: argocd
    app.kubernetes.io/part-of: argocd
  name: argocd-argocd-argocd-server
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - get
  - delete
  - patch
- apiGroups:
  - argoproj.io
  resources:
  - applications
  - applicationsets
  verbs:
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - list
- apiGroups:
  - batch
  resources:
  - jobs
  - cronjobs
  - cronjobs/finalizers
  verbs:
  - create
  - update
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    argocds.argoproj.io/name: argocd
    argocds.argoproj.io/namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
    app.kubernetes.io/name: argocd-server
    app.kubernetes.io/part-of: argocd
```

```
  name: argocd-argocd-argocd-server
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: argocd-argocd-argocd-server
subjects:
- kind: ServiceAccount
  name: argocd-argocd-server
  namespace: argocd
```

i If the privileges described in the default cluster roles are excessive, set `spec.defaultClusterScopedRoleDisabled: true` in the ArgoCD object settings. In this case, cluster roles are not created automatically, and you can define the required privilege level for the ServiceAccount used by the Argo CD instance yourself.

i If needed, you can override the access level to cluster resources at the AppProject object level using the `spec.clusterResourceBlacklist` and `spec.clusterResourceWhitelist` parameters.

Using a custom cluster domain

If the cluster uses a domain other than `cluster.local`, specify it in the `spec.clusterDomain` parameter of the ArgoCD object.

Example:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  clusterDomain: prod.local
```

Additional resources

- Official Argo CD website
- Official Argo CD Operator website
- Module `operator-argo` documentation

Configuring authentication and authorization

Argo CD supports local authentication and is also integrated with the Deckhouse Kubernetes Platform identity and access subsystem. You can learn more about configuring authentication and authorization in the DKP documentation.

⚠ If no additional settings are defined in the ArgoCD object, the local `admin` account with the `admin` role is active by default.

You can learn about managing user and group permissions in the Configuring role-based access control section.

Local authentication

When an ArgoCD object is created, the `admin` user with the `admin` role is created automatically. The `admin` user password is generated automatically. To retrieve it, run:

```
d8 k -n argocd get secret argocd-cluster -o jsonpath='{.data.admin\.password}' | base64 -d
```

Creating additional local users

When creating a local user, you can define whether the user has access to the Argo CD web interface (the `login` attribute) and/or to the Argo CD API (the `apiKey` attribute).

To create local users, set the `spec.localUsers` parameter of ArgoCD, for example:

```
d8 k create -f - <<EOF
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  localUsers:
    # The "deploy" user with rights to sign in to the web interface and generate tokens.
    - name: deploy
      apiKey: true
      login: true
      enabled: true
    # The "ci-bot" user for automation only, without web interface access.
    - name: ci-bot
      apiKey: true
      login: false
      enabled: true
EOF
```

After creating the user and applying the ArgoCD object manifest, generate a password for web interface access if the user has been granted the corresponding rights.

To set a password through a Secret, first calculate its bcrypt hash, then encode the result in Base64:

```
ARGOCD_USER=<USER_NAME>
ARGOCD_PASS=$(echo -n '<USER_PASSWORD>' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' |
base64 -w0)
d8 k -n argocd patch secret argocd-secret -p "{\"data\":
{\"accounts.$ARGOCD_USER.password\":\"$ARGOCD_PASS\"}}"
```

> **i** You can set or change a local user password with the `argocd` CLI utility:
>
> ```
> argocd login <ARGOCD_DOMAIN>:443 --username admin --password <ADMIN_PASSWORD>
> argocd account update-password \
>   --account <ACCOUNT> \
>   --current-password <ADMIN_PASSWORD> \
>   --new-password <NEW_PASSWORD>
> ```

Creating a user token

To work with the API, the user must have the corresponding rights (the `apiKey` parameter; see the example in the section above). To issue a token, use the Argo CD web interface or the `argocd` CLI utility.

To issue a token through the web interface, go to “Settings” → “Accounts”, select the required user, and click “Generate New” in the “Tokens” section.

To issue a token with the `argocd` CLI utility, run the following commands (provide the required values):

```
argocd login <ARGOCD_DOMAIN>:443 --username admin
argocd account generate-token --account <ACCOUNT>
```

> **i** If the user does not have web interface access rights (`login`), another user with administrator rights can issue a token for them.

Disabling local authentication

To disable local authentication, disable the `admin` user by setting the `spec.disableAdmin` parameter of ArgoCD to `true`. Also remove all additional local users if they were created earlier (for details, see Creating additional local users).

After disabling local authentication, review the access rules in the Configuring role-based access control section.

SSO authentication

Before configuring the ArgoCD object, create a DexClient object. It is used as an OAuth2 client and is required for integration with Deckhouse Kubernetes Platform:

```
d8 k create -f -<<EOF
apiVersion: deckhouse.io/v1
kind: DexClient
metadata:
  name: argocd
  namespace: argocd
spec:
  redirectURIs:
    - https://<ARGOCD_DOMAIN>/api/dex/callback
    - https://<ARGOCD_DOMAIN>/api/dex/callback-reserve
EOF
```

`<ARGOCD_DOMAIN>` is the fully qualified domain name (FQDN) set in the `.spec.server.host` section of ArgoCD.

Wait until Deckhouse Kubernetes Platform creates a Secret with the client secret key:

```
d8 k -n argocd get secret/dex-client-argocd
```

Configure the ArgoCD object to use SSO in Deckhouse Kubernetes Platform:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  rbac:
    scopes: "[groups, email]"
  sso:
    dex:
      config: |
        connectors:
          - type: oidc
            id: deckhouse
            name: deckhouse
            config:
              issuer: "https://dex.<CLUSTER_DOMAIN>/"
              clientID: "dex-client-argocd@argocd"
              clientSecret: "$dex-client-argocd:clientSecret"
              insecureEnableGroups: true
              scopes:
                - profile
                - email
                - openid
                - groups
    provider: dex
  server:
    host: <ARGOCD_DOMAIN>
    ingress:
      enabled: true
      ingressClassName: <INGRESS_CLASS_NAME>
      tls:
        - hosts:
            - <ARGOCD_DOMAIN>
          secretName: argocd-ingress-tls
    insecure: true
```

Restart the Argo CD server:

```
d8 k -n argocd rollout restart deploy/argocd-server
```

⚠ If you do not restart the Argo CD server, the sign-in attempt fails, and an error message appears in the Argo CD server log (issue argoproj/argo-cd#13526).

Example error message...

```
time="2024-10-16T14:12:59Z" level=warning msg="Failed to verify token: failed to verify token: token
verification failed for all audiences: error for aud \"argo-cd\": Failed to query provider
\"https://argocd.<ARGOCD_DOMAIN>/api/dex\": Get \"https://argocd.<ARGOCD_DOMAIN>/api/dex/.well-
known/openid-configuration\": tls: failed to verify certificate: x509: certificate is valid for
ingress.local, not argocd.<ARGOCD_DOMAIN>, error for aud \"argo-cd-cli\": Failed to query provider
\"https://argocd.<ARGOCD_DOMAIN>/api/dex\": Get \"https://argocd.<ARGOCD_DOMAIN>/api/dex/.well-
known/openid-configuration\": tls: failed to verify certificate: x509: certificate is valid for
ingress.local, not argocd.<ARGOCD_DOMAIN>"
```

Using a self-signed certificate

First obtain the self-signed certificate used by the Deckhouse Kubernetes Platform identity and access subsystem:

```
d8 k -n d8-user-authn get secret ingress-tls -o jsonpath='{.data.tls\.crt}' | base64 -d
```

Then add the obtained certificate to the OIDC connector configuration in the `rootCAs` section of ArgoCD:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  sso:
    dex:
      config: |
        connectors:
          - type: oidc
            id: deckhouse
            name: deckhouse
            config:
              issuer: "https://dex.<CLUSTER_DOMAIN>/"
              rootCAs:
                - |
                  -----BEGIN CERTIFICATE-----
                  <Self-signed certificate obtained in the previous step>
                  -----END CERTIFICATE-----
              clientID: "dex-client-argocd@argocd"
              clientSecret: "$dex-client-argocd:clientSecret"
              insecureEnableGroups: true
              scopes:
                - profile
                - email
                - openid
                - groups
    provider: dex
  server:
    host: <ARGOCD_DOMAIN>
    ingress:
      enabled: true
      tls:
        - hosts:
            - <ARGOCD_DOMAIN>
          secretName: argocd-ingress-tls
    insecure: true
```

Creating a user token

Argo CD does not allow issuing permanent tokens for users authenticated with SSO. At the same time, such users can use the `argocd` CLI utility by specifying the `--sso` flag during authentication:

```
argocd login <ARGOCD_DOMAIN>:443 --sso
```

When this command runs on the administrator workstation, a web browser opens with the Deckhouse Kubernetes Platform authentication form.

> **i** To issue a permanent token, create a local Argo CD user and set the `apiKey` attribute for them. In this case, the user has access rights only to the Argo CD API. For details, see Creating additional local users.

Additional resources

- Official Argo CD website
- Official Argo CD Operator website
- Module `operator-argo` custom resources
- Module `user-authn` custom resources

Configuring role-based access control

Argo CD uses its own role-based access control (RBAC) model, which is not based on the Kubernetes or Deckhouse Kubernetes Platform role model. The Argo CD role model lets you restrict access to resources and operations through its own policies and roles.

Before configuring role-based access control, complete authentication and authorization setup. After that, assign roles to users and groups and set permissions at the level of the entire Argo CD instance or individual projects with an AppProject object.

You can define role-based access control (RBAC) rules in two places:

- globally — in the ArgoCD object;
- at the project level — in the roles of an AppProject object.

Built-in roles

Argo CD has two predefined roles with a set of policies:

- `role:readonly` — read-only access;
- `role:admin` — access with administrator privileges.

Full description of predefined role policies...

```
# Policies of the readonly role.
p, role:readonly, applications, get, */*, allow
p, role:readonly, applicationsets, get, */*, allow
p, role:readonly, certificates, get, *, allow
p, role:readonly, clusters, get, *, allow
p, role:readonly, repositories, get, *, allow
p, role:readonly, write-repositories, get, *, allow
p, role:readonly, projects, get, *, allow
p, role:readonly, accounts, get, *, allow
p, role:readonly, gpgkeys, get, *, allow
p, role:readonly, logs, get, */*, allow

# Policies of the admin role.
p, role:admin, applications, create, */*, allow
p, role:admin, applications, update, */*, allow
p, role:admin, applications, update/*, */*, allow
p, role:admin, applications, delete, */*, allow
p, role:admin, applications, delete/*, */*, allow
p, role:admin, applications, sync, */*, allow
p, role:admin, applications, override, */*, allow
p, role:admin, applications, action/*, */*, allow
p, role:admin, applicationsets, get, */*, allow
p, role:admin, applicationsets, create, */*, allow
p, role:admin, applicationsets, update, */*, allow
p, role:admin, applicationsets, delete, */*, allow
p, role:admin, certificates, create, *, allow
p, role:admin, certificates, update, *, allow
p, role:admin, certificates, delete, *, allow
p, role:admin, clusters, create, *, allow
p, role:admin, clusters, update, *, allow
p, role:admin, clusters, delete, *, allow
p, role:admin, repositories, create, *, allow
p, role:admin, repositories, update, *, allow
p, role:admin, repositories, delete, *, allow
p, role:admin, write-repositories, create, *, allow
p, role:admin, write-repositories, update, *, allow
p, role:admin, write-repositories, delete, *, allow
p, role:admin, projects, create, *, allow
p, role:admin, projects, update, *, allow
p, role:admin, projects, delete, *, allow
p, role:admin, accounts, update, *, allow
p, role:admin, gpgkeys, create, *, allow
p, role:admin, gpgkeys, delete, *, allow
p, role:admin, exec, create, */*, allow

# The admin role includes all privileges of the readonly role.
g, role:admin, role:readonly

# The local admin user is assigned the admin role.
g, admin, role:admin
```

Default policy for authenticated users

After successful authentication, the user receives the role specified in the `spec.rbac.defaultPolicy` parameter of ArgoCD.

> ⚠ All authenticated users receive at least the permissions defined in the `spec.rbac.defaultPolicy` parameter. These rights cannot be revoked with a `deny` effect rule.

Create a separate role, for example `role:authenticated`, grant it a minimal set of permissions, and use it in `spec.rbac.defaultPolicy`.

Example:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  rbac:
    defaultPolicy: role:authenticated
    policy: |
      p, role:authenticated, applications, get, */*, allow
...
```

Anonymous access

If you enable anonymous access to an Argo CD instance, users can receive rights according to the policy specified in the `spec.rbac.defaultPolicy` parameter without authentication.

Anonymous access is enabled with the `spec.usersAnonymousEnabled` parameter of ArgoCD.

⚠ When enabling anonymous access, create a separate default role, for example `role:unauthenticated`, and assign it in `spec.rbac.defaultPolicy`.

Role-based access control structure

The Argo CD role-based access control syntax is based on the Casbin model and uses two types of records:

- binding a user or group to a role;
- assigning permissions to a role, user, or group.

Role binding

Syntax:

```
g, <user/group>, <role>
```

Where:

- `<user/group>` — a local user, SSO user, or group;
- `<role>` — an internal Argo CD role.

Example:

```
g, my-org:team-beta, role:admin
g, user@example.org, role:readonly
```

Access policy

Syntax:

```
p, <role/user/group>, <resource>, <action>, <object>, <effect>
```

Where:

- `<role/user/group>` — the subject to which the rule is assigned;
- `<resource>` — the resource type;
- `<action>` — the allowed operation;
- `<object>` — the object to which the rule applies;
- `<effect>` — the check result: `allow` or `deny`.

Example:

```
p, role:developer, applications, sync, dev-project/*, allow
p, role:developer, logs, get, dev-project/*, allow
```

i To assign rules to a group, first bind the group to a role with a `g, <group>, <role>` record. After that, assign permissions to the role itself.

Resources and supported actions

The following are the main Argo CD resources and actions that you can use in RBAC policies.

Resource	get	create	update	delete	sync	action	override	invoke
`applications`	✓	✓	✓	✓	✓	✓	✓	✓
`applicationsets`	✓	✓	✓	✓	✓	✓	✓	✓
`clusters`	✓	✓	✓	✓	✓	✓	✓	✓

Resource	get	create	update	delete	sync	action	override	invoke
`projects`	✓	✓	✓	✓	✓	✓	✓	✓
`repositories`	✓	✓	✓	✓	✓	✓	✓	✓
`accounts`	✓	✓	✓	✓	✓	✓	✓	✓
`certificates`	✓	✓	✓	✓	✓	✓	✓	✓
`gpgkeys`	✓	✓	✓	✓	✓	✓	✓	✓
`logs`	✓	✓	✓	✓	✓	✓	✓	✓
`exec`	✓	✓	✓	✓	✓	✓	✓	✓
`extensions`	✓	✓	✓	✓	✓	✓	✓	✓

Application-bound policies

Some resources are bound to specific applications:

- `applications`;
- `applicationsets`;
- `logs`;
- `exec`.

For such resources, the `<object>` value usually has the `<APP_PROJECT>/<APP_NAME>` format.

Example:

```
p, example-user, applications, get, *, allow
p, example-user, logs, get, example-project/my-app, allow
```

If Argo CD has application placement in arbitrary namespaces enabled, the `<object>` format changes to `<APP_PROJECT>/<APP_NAMESPACE>/<APP_NAME>`.

Example:

```
p, example-user, applications, get, */app-namespace/*, allow
p, example-user, logs, get, example-project/app-namespace/my-app, allow
```

The `applications` resource

The `applications` resource supports both basic actions and more detailed rules.

Detailed permissions for `update` and `delete`

The `update` and `delete` permissions granted for an application allow changing or deleting the application itself, but not its nested resources.

To allow an operation on application resources, use the format:

```
<action>/<group>/<kind>/<namespace>/<name>
```

For example, to allow a user to delete only Pods in the `prod-app` application:

```
p, example-user, applications, delete/*/Pod/*/*, default/prod-app, allow
```

To allow updating all application resources but not the application itself:

```
p, example-user, applications, update/*, default/prod-app, allow
```

To deny deleting the application but allow deleting Pods:

```
p, example-user, applications, delete, default/prod-app, deny
p, example-user, applications, delete/*/Pod/*/*, default/prod-app, allow
```

To allow updating the application but deny updating any subresources:

```
p, example-user, applications, update, default/prod-app, allow
p, example-user, applications, update/*, default/prod-app, deny
```

⚠ Argo CD does not use the `/` character as a separator when comparing glob patterns. Therefore, always specify the full resource path, that is, use patterns with four `/` characters.

The `action` action

The `action` action corresponds to built-in or custom actions on application resources.

Format of the `<action>` value:

```
action/<group>/<kind>/<action-name>
```

For example:

- `action/extensions/DaemonSet/restart` — an action for `DaemonSet`;
- `action//Pod/maintenance-off` — an action for `Pod` that has no API group.

Policy example:

```
p, example-user, applications, action//Pod/maintenance-off, default/*, allow
p, example-user, applications, action/extensions/DaemonSet/*, default/*, allow
```

To allow all actions on application resources:

```
p, example-user, applications, action/*, default/*, allow
```

The `override` action

The `override` permission allows passing arbitrary manifests or another revision when synchronizing an `Application`.

⚠ The `override` right lets a user effectively change the composition or state of deployed application resources. Grant it only to users who truly need it.

If the `application.sync.requireOverridePrivilegeForRevisionSync: 'true'` parameter is enabled in the `argocd-cm` ConfigMap, passing an arbitrary revision during synchronization also requires the `override` right.

Other resources

The `applicationsets` resource

The `applicationsets` resource also belongs to application-bound policies. The `create` permission for this resource effectively allows creating Application objects through an ApplicationSet object.

Example:

```
p, dev-group, applicationsets, *, dev-project/*, allow
```

With this rule, users from `dev-group` can create an ApplicationSet that manages applications only within the `dev-project` project.

The `logs` resource

The `get` permission for the `logs` resource allows viewing Pod logs of an application through the Argo CD web interface. In meaning, this is equivalent to `d8 k logs`.

The `exec` resource

The `create` permission for the `exec` resource allows opening an `exec` session in an application Pod through the Argo CD web interface. In meaning, this is similar to `d8 k exec`.

The `extensions` resource

The `extensions` resource is used to invoke proxy extensions. Checks for such rules work together with the `applications` resource: the user must have read access to the application from which the request is made.

Example:

```
p, example-user, applications, get, default/*, allow
p, example-user, extensions, invoke, httpbin, allow
```

The `deny` effect

If a policy with the `deny` effect matches a request, access is denied even if there are more specific rules with the `allow` effect.

The order of lines in `spec.rbac.defaultPolicy` does not affect the result.

Policy evaluation order and matching modes

Access checks are performed in two stages:

1. Rules from `spec.rbac.defaultPolicy` are checked.
2. If no decision is found, rules related to the user and their groups are checked.

If access is explicitly allowed or denied by the default policy, further checks are not performed.

Argo CD supports two matching modes for values set in `spec.rbac.policyMatchMode`:

- `glob` — matching by glob patterns;
- `regex` — matching by regular expressions.

Matching in `glob` mode

In `glob` mode, tokens are treated as whole strings without special separators.

Policy example:

```
p, example-user, applications, action/extensions/*, default/*, allow
```

If the `example-user` user invokes the `extensions/DaemonSet/test` action, the following checks are performed:

1. `example-user` matches `example-user`.
2. `applications` matches `applications`.
3. `action/extensions/DaemonSet/test` matches `action/extensions/*`.
4. `default/my-app` matches `default/*`.

Using SSO users and groups

The `scopes` parameter defines which OIDC scopes Argo CD must analyze during RBAC checks in addition to `sub`. If the parameter is not set, the default value is `'[groups]'`.

Example of an ArgoCD object with scopes configured:

```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
  name: argocd
  namespace: argocd
spec:
  rbac:
    policy: |
      p, my-org:team-alpha, applications, sync, my-project/*, allow
      g, my-org:team-beta, role:admin
      g, user@example.org, role:admin
      g, admin, role:admin
      g, role:admin, role:readonly
    defaultPolicy: role:readonly
    scopes: '[groups, email]'
```

In this example:

1. `g, admin, role:admin` explicitly binds the built-in `admin` user to the `role:admin` role.
2. `g, role:admin, role:readonly` sets role inheritance: everyone who has `role:admin` automatically also gets `role:readonly` rights.

You can combine this approach with roles at the AppProject level. For example, you can create the `team-beta-project` project and assign rights to users and groups:

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-beta-project
  namespace: argocd
spec:
  roles:
    - name: admin
      description: Administrator permissions for team-beta.
      policies:
        - p, proj:team-beta-project:admin, applications, *, team-beta-project/*, allow
      groups:
        - user@example.org
        - my-org:team-beta
```

Local users

You can grant access to local users in two ways:

- assign rules directly;
- bind the user to a role.

Example of assigning a rule directly:

```
p, my-local-user, applications, sync, my-project/*, allow
```

Example of binding to a role:

```
g, my-local-user, role:admin
```

> ⚠ If SSO and local users are used at the same time, ambiguity is possible. For example, if the local user `sally` is bound to `role:admin`, and one of the SSO scope values is also `sally`, that SSO user may also receive `role:admin` rights.

To avoid this situation, when using SSO and local users together, assign rules to local users directly rather than through roles.

Example:

```
p, my-local-user, *, *, *, allow
```

Testing RBAC policies

To verify RBAC rules, use the `argocd` CLI utility. To check whether a specific user, group, or role can perform a certain action, use the following command:

```
argocd admin settings rbac can
```

For example, run the command below to check whether the `admin@deckhouse.io` user has rights to create `applications` in the `default` project:

```
argocd admin settings rbac can admin@deckhouse.io create application 'default/*' --namespace argocd
```

Additional resources

- Official Argo CD website
- Official Argo CD Operator website
- Module `operator-argo` custom resources

Marketplace

Marketplace is a system for managing Deckhouse Kubernetes Platform (DKP) delivery units (Packages). It lets administrators connect package registries, discover available packages, and make them available to project users for installation.

> **i** Marketplace is available starting from DKP version 1.76.

Administrator tasks

A cluster administrator do the following tasks:

1. Connect a package registry by creating a PackageRepository resource.
2. Monitor scanning operations that discover packages in the registry.
3. Ensure users can see available package versions and install them into their namespaces.

Users interact with packages through the Application object (read more in the Using → Marketplace section).

Key resources

Resource	Short name	Scope	Description
`PackageRepository`	—	Cluster	Source registry for packages
`PackageRepositoryOperation`	`pro`	Cluster	Scanning operation on a repository
`ApplicationPackageVersion`	`apv`	Cluster	Discovered version of a package
`ApplicationPackage`	—	Cluster	Aggregate metadata for a package
`Application`	`app`	Namespace	Installed application instance (managed by users)

The Package repositories section describes how to connect a registry and check its status.

The Scanning section describes how to monitor scan operations and trigger manual scans.

Additional resources

- Using Marketplace
- Global custom resources

Package repositories

Connecting the Deckhouse Kubernetes Platform (DKP) to a container registry containing application packages is done using the PackageRepository. Once connected, DKP automatically scans the registry and creates ApplicationPackageVersion objects for each discovered package version.

Example of a PackageRepository manifest:

```
apiVersion: deckhouse.io/v1alpha1
kind: PackageRepository
metadata:
  name: my-registry
spec:
  registry:
    repo: registry.example.com/packages
    scheme: HTTPS
    dockerCfg: <base64-encoded-docker-config>
```

Authentication and Scanning Interval Management

Authentication

One of the following methods can be used for authentication in the registry:

- **`dockerCfg`**: base64-encoded Docker config JSON (`~/.docker/config.json` format). Preferred when the registry uses token-based authentication.
- **`login`** + **`password`**: explicit credentials.

```
spec:
  registry:
    repo: registry.example.com/packages
    scheme: HTTPS
    login: my-user
    password: my-password
```

If the registry uses a self-signed TLS certificate, provide it via `ca`:

```
spec:
  registry:
    repo: registry.example.com/packages
    scheme: HTTPS
    dockerCfg: <base64-encoded-docker-config>
    ca: |
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
```

Scan interval

By default, DKP rescans the registry every **6 hours**. The interval can be overridden using the `scanInterval` parameter:

```
spec:
  registry:
    repo: registry.example.com/packages
  scanInterval: 1h30m
```

Checking repository status

The repository's status is displayed in the PackageRepository object's status.

To display brief information about the status, use the following command:

```
d8 k get packagerepository <REPOSITORY_NAME>
```

Output columns:

Column	Description
`Phase`	Current state of the repository
`Scan`	Timestamp of the last scan
`MSG`	Message from the last scan condition
`Packages`	Total number of packages discovered (hidden by default, use `-o wide`)

For detailed information about the status, use the following command:

```
d8 k get packagerepository <REPOSITORY_NAME> -o yaml
```

Key status fields:

Field	Description
`status.phase`	Current repository phase
`status.lastScanTime`	Time of the most recent scan of any outcome
`status.lastChangeTime`	Time of the last scan that found at least one new version
`status.lastNewVersions`	Number of new versions found in the most recent scan
`status.packagesCount`	Total packages in the repository
`status.packages[]`	List of packages with `name` and `type` fields

Field	Description
`status.conditions`	Detailed conditions, including `LastScanSucceeded`

The `LastScanSucceeded` condition:

```
d8 k get packagerepository my-registry \
  -o jsonpath='{.status.conditions[?(@.type=="LastScanSucceeded")].message}'
```

Viewing discovered package versions

After a successful scan, ApplicationPackageVersion objects appear in the cluster (you can use the abbreviated name `apv`):

```
d8 k get apv
```

Example output:

```
NAME                           PACKAGE     REPOSITORY   TRANSITIONTIME   METADATALOADED
MESSAGE   USEDBY
my-registry-redis-v7.2.0       redis       my-registry  5m               True
my-registry-postgres-v15.0.0   postgres    my-registry  5m               True
```

Filtering by package name can be done using the following command (in this example, versions of the `redis` package are being filtered):

```
d8 k get apv -l package=redis
```

> **i** `MetadataLoaded=True` means the package's OpenAPI schema, description, and requirements were successfully loaded from the registry. A package with `MetadataLoaded=False` cannot be installed until the metadata is retrieved.

Additional resources

- Global custom resources

Scanning

Deckhouse Kubernetes Platform (DKP) uses PackageRepositoryOperation to scan package registries. Each scan operation discovers new package versions and creates or updates ApplicationPackageVersion objects. Operations are created automatically on a schedule or can be created manually when needed.

Viewing scan operations

To view all scanning operations, use the command (you can use `pro` as a shorthand for `packagerepositoryoperations`):

```
d8 k get pro
```

Example output:

```
NAME                    COUNT   COMPLETED   MSG   COMPLETIONTIME
test-scan-1780052895    23      True              3h38m
test-scan-1780053890    23      True              3h22m
```

Output columns:

Column	Description
`Count`	Total number of packages found during the scan
`Completed`	Whether the operation finished (`True` / `False`)
`MSG`	Message from the `Completed` condition
`CompletionTime`	Time when the operation completed

To filter operations by a specific repository, use the following command:

```
d8 k get pro -l packages.deckhouse.io/repository=<REPO_NAME>
```

Inspecting a scan operation

For full scan results including per-package details, use the following command:

```
d8 k get pro <operation-name> -o yaml
```

Key status fields:

Field	Description
`status.startTime`	When the operation started
`status.completionTime`	When the operation completed
`status.packages.total`	Total packages found
`status.packages.processedOverall`	Total packages successfully processed
`status.packages.newVersionsOverall`	Total new versions across all packages
`status.packages.processed[]`	Per-package results: `name`, `type`, `foundVersions`, `newVersions`
`status.packages.failed[]`	Packages with errors: `name`, `errors[]` with `version` and `message`
`status.packages.discovered[]`	Packages first seen in this operation

An example command for viewing packages with errors:

```
d8 k get pro <operation-name> \
  -o jsonpath='{range .status.packages.failed[*]}{.name}: {range .errors[*]}{.version} -
{.message}{"\n"}{end}{end}'
```

Triggering a manual scan

By default, each PackageRepository creates a new scan operation **6 hours** after the previous one completed (configurable via `spec.scanInterval`).

To scan immediately, create a PackageRepositoryOperation manually with the following example:

```
apiVersion: deckhouse.io/v1alpha1
kind: PackageRepositoryOperation
spec:
  packageRepositoryName: my-registry
  type: Update
  update:
    fullScan: true
    timeout: 5m
```

> **i** Use `generateName` instead of `name` so you can create multiple operations without name conflicts.

Alternatively, to create a scan operation, you can use the following command:

```
d8 system package scan <REPO_NAME>
```

This command creates a PackageRepositoryOperation with `spec.type: Update` and `spec.update.fullScan: true`.

`fullScan` flag

Value	Behavior
`true`	Re-checks all tags in the registry, including those already known
`false` (default)	Only processes tags added since the last scan (incremental)

Use `fullScan: true` when you suspect the registry was modified outside the normal workflow, or when ApplicationPackageVersion objects are missing versions that exist in the registry.

Additional resources

- Global custom resources

Platform update

Deckhouse Kubernetes Platform (DKP) includes a built-in update management mechanism for both the platform itself and Kubernetes cluster components. To roll out new versions consistently and safely, DKP uses a five-channel update system ranging from the newest and unstable (Alpha) to the most thoroughly tested (Rock Solid). Each new version gradually moves through these channels, helping identify issues early and ensuring stability in production environments. For details on each channel, refer to the Architecture section.

⚠ DKP does not support version downgrades. After a release is installed, rolling the platform back to an earlier minor or patch version is not possible.

When switching to a more stable channel, the platform remains on the current release until a newer version becomes available on that channel.

Rolling back by one minor version is possible only for the Kubernetes (control plane) version, not for the DKP version.

You can configure the update process. Both automatic and manual modes are supported, as well as update windows and other parameters. See Update configuration for details.

An alerting system is used to track the update status and quickly report any issues. For more information, refer to Notification settings.

The following features are supported:

- Release notifications: DKP can send release notifications via webhooks.
- Retrieving the changelog: Each DKP release includes a changelog, available both in the cluster and as part of the release notification.
- Checking dependencies before update: Checks for component dependencies before proceeding with an update to prevent conflicts.

For more details on updating DKP in air-gapped environment, see the “Updating DKP in air-gapped environment” guide.

Configuring updates

Deckhouse Kubernetes Platform (DKP) supports a flexible update mechanism, allowing you to select release channels and configure the update mode. Release channels help you balance stability with the speed of receiving new features.

The update mode configuration lets you choose between automatic or manual updates and define update windows during which new versions can be installed. Together, these features help you avoid updates at inconvenient times and control migration to new releases.

i Up-to-date information about DKP versions available on different release channels is available at releases.deckhouse.io.

Checking the current release channel

To check which release channel is used in your cluster, run the following command:

```
d8 k get mc deckhouse -o yaml | grep releaseChannel
```

Example output:

```
    releaseChannel: Stable
```

Switching release channels

To switch the release channel, specify the new channel in the `settings.releaseChannel` parameter of the `deckhouse` module.

Example configuration using the `Stable` channel:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: Stable
```

Update modes

DKP supports two update modes that determine how new versions are applied:

- **Automatic mode** — the cluster updates as soon as a new version appears on the selected release channel. Two configuration options are available:
  - `AutoPatch` — only patch version updates within the current minor version are applied;
  - `Auto` — both patch and minor versions are applied automatically.
- **Manual mode (`Manual`)** — both patch and minor versions are applied only after manual approval.

For automatic mode, update windows settings are available — allowed updates are applied taking `update.windows` into account, if windows are configured. If no windows are set, the update is applied as soon as the version appears on the release channel.

> **i**
>
> The `update` settings of the `deckhouse` module (mode and windows) apply by default to both built-in and external modules.
>
> If a module has no dedicated update policy (ModuleUpdatePolicy) and its ModuleConfig does not specify the `updatePolicy` parameter, the module inherits `releaseChannel` and `update` from the `deckhouse` ModuleConfig.

Patch versions only within the current minor (AutoPatch)

To keep DKP updates within the current minor version (apply patch versions only), use the `AutoPatch` mode.

For example, if version `v1.70.1` is installed, DKP can automatically update to `v1.70.2`, but will not move to `v1.71.*` without manual approval.

This is the default value. To set the `AutoPatch` mode explicitly, run the following command:

```
d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"AutoPatch"}}}}'
```

To approve a minor version update, run the following command, replacing `<DECKHOUSE-VERSION>` with the target DKP version:

```
d8 k patch DeckhouseRelease <DECKHOUSE-VERSION> --type=merge -p='{"approved": true}'
```

Automatic updates for all versions (Auto)

In the `Auto` mode, DKP automatically applies both patch and minor versions, taking update windows into account if they are configured.

To enable the `Auto` mode, the following command:

```
d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"Auto"}}}}'
```

Manual mode (Manual)

In the `Manual` mode, DKP receives information about new versions in the cluster, but applying both patch and minor versions requires manual approval.

To enable the `Manual` mode, the following command:

```
d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"Manual"}}}}'
```

Checking the current update mode

To determine the current update mode used in the cluster, inspect the configuration of the `deckhouse` module with the following command:

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

Example output:

```
spec:
  settings:
    releaseChannel: Stable
    update:
      mode: AutoPatch
      windows:
      - days:
        - Mon
        from: "19:00"
        to: "20:00"
```

How automatic updates are applied

Automatic update mode is enabled when the `releaseChannel` parameter is specified in the `deckhouse` module configuration. When this condition is met:

1. DKP checks the release channel every minute for new releases.
2. When a new release appears, DKP downloads it into the cluster and creates a DeckhouseRelease custom resource.
3. Once the DeckhouseRelease resource appears in the cluster, DKP applies the corresponding update according to the configured update settings (by default — automatically, at any time).

To view the list and status of all releases in the cluster, run the following command:

```
d8 k get deckhousereleases
```

⚠ Starting from DKP 1.70, patch version updates (for example, an update to version `1.70.2` when version `1.70.1` is installed) are applied taking update windows into account. Prior to DKP 1.70, patch version updates are applied without regard to the update mode and windows.

Release pinning

Release pinning refers to fully or partially disabling automatic updates.

There are three ways to restrict automatic updates in Deckhouse:

- Enable manual update approval mode.

  In this mode, DKP will receive updates into the cluster, but applying patch and minor versions will require manual approval.

  To enable manual update approval mode, set the `settings.update.mode` parameter to `Manual` in the `deckhouse` module configuration using the following command:

  ```
  d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"Manual"}}}}'
  ```

  To approve an update, run the following command, replacing `<DECKHOUSE-VERSION>` with the target DKP version:

  ```
  d8 k patch DeckhouseRelease <DECKHOUSE-VERSION> --type=merge -p='{"approved": true}'
  ```

- Enable automatic updates for patch versions only.

  In this mode, DKP will receive updates into the cluster, but applying minor versions will require manual approval. Patch versions within the current minor version will be applied automatically, taking update windows into account if they are configured.

  For example, if you have DKP version `v1.70.1` installed, after enabling this mode, Deckhouse can automatically update to `v1.70.2`, but it will not update to `v1.71.*` without manual approval.

  To enable automatic updates for patch versions only, set the `settings.update.mode` parameter to `AutoPatch` in the `deckhouse` module configuration using the following command:

  ```
  d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":
  {"mode":"AutoPatch"}}}}'
  ```

To approve a minor version update, run the following command, replacing `<DECKHOUSE-VERSION>` with the target DKP version:

```
d8 k patch DeckhouseRelease <DECKHOUSE-VERSION> --type=merge -p='{"approved": true}'
```

- Manually set the target DKP version tag for the `deckhouse` Deployment and remove the `releaseChannel` parameter from the `deckhouse` module configuration.

In this case, DKP will remain at the specified version, and no information about newer available versions (DeckhouseRelease objects) will appear in the cluster.

> **i** **Important**. This mode blocks the installation of patch releases, which may include critical security or bug fixes. If you need to receive patches within the current minor version, use the `AutoPatch` mode instead of hard pinning.

Example of pinning version `v1.66.3` for DKP EE and removing the `releaseChannel` parameter from the `deckhouse` module configuration:

```
d8 k -ti -n d8-system exec svc/deckhouse-leader -c deckhouse -- kubectl set image deployment/deckhouse
deckhouse=registry.deckhouse.io/deckhouse/ee:v1.66.3
```

```
d8 k patch mc deckhouse --type=json -p='[{"op": "remove", "path":
"/spec/settings/releaseChannel"}]'
```

Manual update approval

Manual approval of DKP updates is required in the following cases:

- The DKP update confirmation mode is enabled.

  This means the `settings.update.mode` parameter of the `deckhouse` module is set to either `Manual` (confirmation required for both patch and minor updates) or `AutoPatch` (confirmation required only for minor updates).

  To approve an update, run the following command, replacing `<DECKHOUSE-VERSION>` with the target version:

  ```
  d8 k patch DeckhouseRelease <DECKHOUSE-VERSION> --type=merge -p='{"approved": true}'
  ```

- Automatic update approval is disabled for a NodeGroup, for updates that might cause temporary downtime of system components.

  This means the `spec.disruptions.approvalMode` parameter of the corresponding NodeGroup resource is set to `Manual`.

  To apply the update, set the `update.node.deckhouse.io/disruption-approved=` annotation on each node in the group:

  Example:

  ```
  d8 k annotate node ${NODE_1} update.node.deckhouse.io/disruption-approved=
  ```

Update windows

DKP allows you to define *update windows*, which are specific time intervals during which automatic updates are allowed. Using update windows ensures that updates won't be installed at inconvenient times or during periods of high cluster load.

Applying updates when update windows are configured

- If update windows are configured, DKP installs new versions only during the specified windows.
- If no update windows are configured, the update is applied as soon as a new version appears on the configured release channel.

Configuring update windows

You can manage DKP update windows in the following ways:

- **To control general updates**, use the `update.windows` parameter in the `deckhouse` module configuration;
- **To control updates that may lead to short-term downtime of system components**, use the

`disruptions.automatic.windows` and `disruptions.rollingUpdate.windows` parameters in the NodeGroup resource.

Configuration examples

- Two daily update windows: from 08:00 to 10:00 and from 20:00 to 22:00 (UTC):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: EarlyAccess
    update:
      windows:
        - from: "8:00"
          to: "10:00"
        - from: "20:00"
          to: "22:00"
```

- Update windows on Tuesdays and Saturdays from 18:00 to 19:30 (UTC):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: Stable
    update:
      windows:
        - from: "18:00"
          to: "19:30"
          days:
            - Tue
            - Sat
```

Additional resources

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

Configuring notifications on new releases

Deckhouse Kubernetes Platform (DKP) generates alerts in the monitoring system and can automatically send notifications about upcoming minor updates to external systems. This helps you plan updates and prepare for them in advance.

Conditions for sending notifications to external systems:

- DKP is operating in automatic update mode.
- A minor version update is planned (notifications are not sent for patch versions).
- A webhook for notifications is configured.

Alerts in the monitoring system

If an update requires making changes to the cluster (for example, updating the Kubernetes or OS version), DKP generates designated alerts. These alerts include:

- `D8NodeHasDeprecatedOSVersion`: Nodes with an unsupported OS version have been detected in the cluster.
- `HelmReleasesHasResourcesWithDeprecatedVersions`: Some Helm releases use deprecated resources.
- `D8KubernetesVersionIsDeprecated`: The installed Kubernetes version is no longer supported.

If any of these alerts appear, make sure to resolve them before updating. This helps avoid disruptions and ensures the cluster remains stable after the update.

Configuring notifications

In the `Auto` update mode, you can configure a webhook call to receive a notification about an upcoming minor DKP version update.

Additionally, notifications are generated not only for Deckhouse updates but also for updates of any modules, including individual ones.
In some cases, the system may initiate multiple notifications simultaneously (10–20 notifications) at approximately 15-second intervals.

> **i** Notifications are available only in the `Auto` update mode; they are not generated in the `Manual` mode.

> **i** Specifying a webhook is optional: if the `update.notification.webhook` parameter is not set but the `update.notification.minimalNotificationTime` parameter is specified, the update will still be postponed for the defined duration. In this case, the appearance of the DeckhouseRelease resource in the cluster, named after the new version, can be considered the notification.

After a new minor DKP version appears in the selected update channel but before it is applied in the cluster, a POST request will be sent to the configured webhook address.

The minimalNotificationTime parameter allows postponing the update installation for a defined period, providing time to react to the notification while respecting update windows.
If the webhook is unavailable, each failed attempt to send the notification will postpone the update by the same amount, which may lead to the update being indefinitely deferred.

> **⚠** If your webhook returns any status code out of 2xx range, DKP retries sending the notification up to five times with exponential backoff. If all attempts fail, the release is blocked until the webhook becomes available again.

For easier error handling and debugging, when returning error codes the webhook should return a JSON response with the following structure:

- `code` — optional internal error code for programmatic handling;
- `message` — a human-readable description of what went wrong.

If the webhook returns a successful HTTP status (2xx), DKP treats the notification as successful regardless of the response body.

Minimal Go webhook example...

```
package main
import (
  "encoding/json"
  "fmt"
  "log"
  "net/http"
)
// Payload structure Deckhouse sends in POST body.
type WebhookData struct {
  Subject      string          `json:"subject"`
  ModuleName   string          `json:"moduleName,omitempty"`
  Version      string          `json:"version"`
  Requirements  map[string]string `json:"requirements,omitempty"`
  ChangelogLink string          `json:"changelogLink,omitempty"`
  ApplyTime    string          `json:"applyTime,omitempty"`
  Message      string          `json:"message"`
}

// Response structure that Deckhouse expects from webhook on error
type ResponseError struct {
  Code    string `json:"code,omitempty"`
  Message string `json:"message"`
}

func handler(w http.ResponseWriter, r *http.Request) {
  if r.Method != http.MethodPost {
    w.WriteHeader(http.StatusMethodNotAllowed)
    return
```

```
  }
  defer r.Body.Close()

  var data WebhookData
  if err := json.NewDecoder(r.Body).Decode(&data); err != nil {
    log.Printf("failed to decode payload: %v", err)
    w.WriteHeader(http.StatusInternalServerError)
    return
  }

  // Print payload fields
  log.Printf("subject=%s version=%s applyTime=%s changelog=%s requirements=%v",
    data.Subject, data.Version, data.ApplyTime, data.ChangelogLink, data.Requirements)
  log.Printf("message=%s", data.Message)

  // Example conditional logic: fail intentionally for testing
  if data.Version == "v0.0.0-fail" {
    // Return structured error response with error status code
    errorResp := ResponseError{
      Code:   "TEST_FAILURE",
      Message: "intentional failure for testing",
    }

    w.WriteHeader(http.StatusBadRequest)
    json.NewEncoder(w).Encode(errorResp)
    return
  }

  // Return success response with 2xx status code
  w.WriteHeader(http.StatusOK)
  w.Write([]byte("Notification processed successfully"))
}

func main() {
  mux := http.NewServeMux()
  mux.HandleFunc("/webhook", handler)

  addr := ":8080"
  fmt.Printf("listening on %s, POST to http://localhost%s/webhook\n", addr, addr)
  if err := http.ListenAndServe(addr, mux); err != nil {
    log.Fatal(err)
  }
}
```

Supported notifications parameters

- `update.notification.webhook`: URL for sending notifications. A POST request with update information is sent as soon as a new minor version appears on the selected release channel, but before the update is installed in the cluster.
- `update.notification.auth`: Authentication parameters for calling the webhook. If this parameter is not set, authentication is not used.
  - `update.notification.auth.basic`: Basic authentication. The username and password are passed in the `Authorization` header as `Basic <base64(username:password)>`.
    - `update.notification.auth.basic.username`: Username.
    - `update.notification.auth.basic.password`: Password.
  - `update.notification.auth.bearerToken`: Token-based authentication. The token is passed in the `Authorization` header as `Bearer <token>`.
- `update.notification.minimalNotificationTime`: Minimum interval between the appearance of a new minor version on the selected release channel and the start of the update. Specified in hours and minutes: `30m`, `1h`, `2h30m`, `24h`. If an update window is configured, the update will be applied only after the `minimalNotificationTime` has elapsed and within the defined window.
- `update.notification.tlsSkipVerify`: Disables TLS certificate verification when calling the webhook (for example, if a self-signed certificate is used). Set to `false` by default.

Example `update.notification` configuration using basic authentication:

```
update:
  notification:
    webhook: https://release-webhook.mydomain.com
    minimalNotificationTime: 4h
    auth:
      basic:
        username: myusername
        password: mypassword
    tlsSkipVerify: true
```

Example `update.notification` configuration without authentication:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    update:
      releaseChannel: Stable
      mode: Auto
      notification:
        webhook: https://release-webhook.mydomain.com
        minimalNotificationTime: 8h
```

Notification format

When the conditions for sending notifications are met, DKP sends a POST request to the specified webhook with the header `Content-Type: application/json`.

Example request body for Deckhouse release:

```
{
  "subject": "Deckhouse",
  "version": "1.68",
  "requirements":  {"k8s": "1.29.0"},
  "changelogLink": "https://github.com/deckhouse/deckhouse/blob/main/CHANGELOG/CHANGELOG-
v1.68.md",
  "applyTime": "2025-02-01T14:30:00Z00:00",
  "message": "New Deckhouse Release 1.68 is available. Release will be applied at:
Wednesday, 05-Feb-25 14:30:00 UTC"
}
```

Example request body for module release:

```
{
  "subject": "Module",
  "moduleName": "test",
  "version": "0.7.14",
  "requirements": {"deckhouse": ">= 1.69"},
  "changelogLink": "https://github.com/deckhouse/modules/blob/main/CHANGELOG/CHANGELOG-
test-v0.7.14.md",
  "applyTime": "2025-12-17T12:32:24Z",
  "message": "New test Release 0.7.14 is available. Release will be applied at: Wednesday,
17-Dec-25 12:32:24 UTC"
}
```

Field descriptions:

- `subject`: Release subject (“Deckhouse” for Deckhouse releases or “Module” for module releases).
- `moduleName`: Module name (present only for module releases).
- `version`: Minor version number (a string).
- `requirements`: Object with requirements for the new version (for example, the minimum Kubernetes version).
- `changelogLink`: Link to the changelog describing changes in the new minor version.
- `applyTime`: Scheduled update date and time in RFC3339 format with the configured update windows

considered.
- `message`: A short text notification about the available minor version and its scheduled installation time.

Additional resources

- Global custom resources
- Module `deckhouse` parameters

Updating Kubernetes and versioning

Updating and version management

The control plane update process in DKP is fully automated.

- DKP supports the latest five Kubernetes versions.
- You can roll back the control plane one minor version and upgrade forward several minor versions — one at a time.
- Patch versions (e.g., `1.27.3` → `1.27.5`) are updated automatically with Deckhouse and cannot be managed manually.
- Minor versions are set manually using the `kubernetesVersion` parameter in the ClusterConfiguration resource.

Changing the Kubernetes version

You can downgrade only one minor version from the highest version ever used in the cluster. Upgrades proceed one minor version at a time.

1. Open the ClusterConfiguration editor:

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

2. Set the target Kubernetes version using the `kubernetesVersion` field:

   ```
   apiVersion: deckhouse.io/v1
   kind: ClusterConfiguration
   cloud:
     prefix: demo-stand
     provider: Yandex
   clusterDomain: cloud.education
   clusterType: Cloud
   defaultCRI: Containerd
   kubernetesVersion: "1.30"
   podSubnetCIDR: 10.111.0.0/16
   podSubnetNodeCIDRPrefix: "24"
   serviceSubnetCIDR: 10.222.0.0/16
   ```

3. Save the changes.
4. Wait for the update to complete. You can track the update progress with the `d8 k get no` command. The update can be considered complete when the updated version appears in the `VERSION` column of each cluster node in the command output.

Monitoring Kubernetes update progress

The `control-plane-manager` module includes the `update-observer` component, which provides up-to-date information about the Kubernetes version update process in the cluster.

`update-observer` component:

- reads cluster configuration from the `d8-cluster-configuration` Secret
- tracks kubelet versions on all nodes via `nodeInfo.kubeletVersion`
- collects versions from all control plane instances via the `control-plane-manager.deckhouse.io/kubernetes-version` annotation
- creates and maintains the **`d8-cluster-kubernetes`** ConfigMap in the `kube-system` namespace with detailed update status.

The `d8-cluster-kubernetes` ConfigMap displays:

- **Component status**: Versions of control plane components (kube-apiserver, kube-scheduler, kube-controller-manager) on each master node.
- **Node progress**: How many nodes have been updated and the total count.
- **Target and current version**: The desired version from configuration and the actual state during the update.
- **Version mismatch**: If any components are running a version different from the target (including newer than desired).

- **Version lists**:
  - `supportedVersions`: Lists minor Kubernetes versions supported in the current DKP release.
  - `availableVersions`: Lists versions that can be selected for upgrade or downgrade in the *current* cluster. The set is limited by the highest minor version ever installed on the cluster and by the rule that downgrade proceeds one minor at a time.
  - `automaticVersion`: Minor version that will be used when the update mode is `Automatic`.

During `ControlPlaneUpdating`, `status.progress` reflects overall upgrade progress across intermediate minor versions. For a multi-hop upgrade (for example, 1.33 → 1.35), the percentage increases as each hop completes, not only when every control plane component reaches the final target.

Minor versions in the ConfigMap (`spec`, `status`, and metadata labels such as `k8s-version` and `max-k8s-version`) use the same string format as in ClusterConfiguration, meaning without a `v` prefix (for example, `"1.33"`).

You can see in real time which components are being updated, at what stage the process is, and whether the update has “stuck” on any node or component.

To view the update status, run the command:

```
d8 k get configmap d8-cluster-kubernetes -n kube-system -o yaml
```

ConfigMap content examples

The `data.spec` and `data.status` fields store YAML with the `spec` field (target version and update mode) and the `status` field (current state). Below are examples of the content for various situations.

Cluster up to date (3 master nodes, 3 worker nodes)

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.32"
    updateMode: Manual
  status: |
    currentVersion: "1.32"
    supportedVersions:
    - "1.30"
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    availableVersions:
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    automaticVersion: "1.33"
    phase: UpToDate
    controlPlane:
    - name: master-1
      phase: UpToDate
      components:
        kube-apiserver: "1.32"
        kube-controller-manager: "1.32"
        kube-scheduler: "1.32"
    - name: master-2
      phase: UpToDate
      components:
        kube-apiserver: "1.32"
        kube-controller-manager: "1.32"
        kube-scheduler: "1.32"
    - name: master-0
      phase: UpToDate
      components:
        kube-apiserver: "1.32"
        kube-controller-manager: "1.32"
        kube-scheduler: "1.32"
    nodes:
      desiredCount: 6
      upToDateCount: 6
kind: ConfigMap
metadata:
  annotations:
    cause: idle
    lastReconciliationTime: "2026-02-02T01:13:05Z"
    lastUpToDateTime: "2026-01-30T16:26:36Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.32"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "20837731"
  uid: ba981996-f737-469c-9ce1-53aa46135994
```


Start of update (e.g., Kubernetes version downgrade)

The target version is already set; the control plane or nodes are still transitioning to it.

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.32"
    updateMode: Manual
  status: |
    currentVersion: "1.33"
    supportedVersions:
    - "1.30"
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    availableVersions:
    - "1.32"
    - "1.33"
    - "1.34"
    automaticVersion: "1.33"
    phase: ControlPlaneUpdating
    progress: 0%
    controlPlane:
    - name: master-0
      phase: Updating
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    - name: master-1
      phase: Updating
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    - name: master-2
      phase: Updating
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    nodes:
      desiredCount: 6
      upToDateCount: 0
kind: ConfigMap
metadata:
  annotations:
    cause: downgradeK8s
    lastReconciliationTime: "2026-02-02T11:34:42Z"
    lastUpToDateTime: "2026-02-02T11:09:59Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.33"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "21379847"
  uid: ba981996-f737-469c-9ce1-53aa46135994
```

Control plane update in progress

Some master nodes are already on the new version, others are still updating; progress is shown as a percentage:

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.32"
    updateMode: Manual
  status: |
    currentVersion: "1.33"
    supportedVersions:
    - "1.30"
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    availableVersions:
    - "1.32"
    - "1.33"
    - "1.34"
    automaticVersion: "1.33"
    phase: ControlPlaneUpdating
    progress: 60%
    controlPlane:
    - name: master-0
      phase: Updating
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    - name: master-1
      phase: Updating
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    - name: master-2
      phase: UpToDate
      components:
        kube-apiserver: "1.32"
        kube-controller-manager: "1.32"
        kube-scheduler: "1.32"
    nodes:
      desiredCount: 6
      upToDateCount: 6
kind: ConfigMap
metadata:
  annotations:
    cause: downgradeK8s
    lastReconciliationTime: "2026-02-02T11:41:55Z"
    lastUpToDateTime: "2026-02-02T11:09:59Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.33"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "21388343"
  uid: ba981996-f737-469c-9ce1-53aa46135994
```

Intermediate step of a multi-hop upgrade (e.g. 1.33 → 1.35)

In the configuration, the target Kubernetes version may be several minor versions ahead of the current cluster version. In this case, the upgrade is performed step by step through intermediate minor versions.

The `status.currentVersion` field reflects the current upgrade step (the active minor version the cluster is being brought to at the moment). Within a single step, individual control plane components may temporarily run different minor versions.

The `progress` field reflects the entire upgrade path, including intermediate versions. Therefore, its value can be significantly greater than 0% even before all components reach the target version.

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.35"
    updateMode: Manual
  status: |
    currentVersion: "1.34"
    supportedVersions:
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    - "1.35"
    availableVersions:
    - "1.33"
    - "1.34"
    - "1.35"
    automaticVersion: "1.33"
    phase: ControlPlaneUpdating
    progress: 60%
    controlPlane:
    - name: master-0
      phase: Updating
      components:
        kube-apiserver: "1.35"
        kube-controller-manager: "1.34"
        kube-scheduler: "1.34"
    - name: master-1
      phase: Updating
      components:
        kube-apiserver: "1.34"
        kube-controller-manager: "1.34"
        kube-scheduler: "1.34"
    - name: master-2
      phase: Updating
      components:
        kube-apiserver: "1.34"
        kube-controller-manager: "1.34"
        kube-scheduler: "1.34"
    nodes:
      desiredCount: 6
      upToDateCount: 0
kind: ConfigMap
metadata:
  annotations:
    cause: upgradeK8s
    lastReconciliationTime: "2026-04-17T09:43:08Z"
    lastUpToDateTime: "2026-04-15T13:43:59Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.33"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "1753458"
  uid: ba981996-f737-469c-9ce1-53aa46135994
```

Cluster up to date (2 master nodes, 1 arbitr node and 3 worker nodes)

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.33"
    updateMode: Manual
  status: |
    currentVersion: "1.33"
    supportedVersions:
    - "1.30"
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    availableVersions:
    - "1.32"
    - "1.33"
    - "1.34"
    automaticVersion: "1.33"
    phase: UpToDate
    controlPlane:
    - name: master-0
      phase: UpToDate
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    - name: master-1
      phase: UpToDate
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    nodes:
      desiredCount: 6
      upToDateCount: 6
kind: ConfigMap
metadata:
  annotations:
    cause: upgradeK8s
    lastReconciliationTime: "2026-02-02T11:09:59Z"
    lastUpToDateTime: "2026-02-02T11:09:59Z"
  creationTimestamp: "2026-01-16T16:48:45Z"
  labels:
    heritage: deckhouse
    k8s-version: "1.33"
    max-k8s-version: "1.33"
  name: d8-cluster-kubernetes
  namespace: kube-system
  resourceVersion: "21357074"
  uid: ba981996-f737-469c-9ce1-53aa46135994
```

Failure of one or more control plane components

The master node has `phase: Failed`; the `description` field contains the reason (e.g., pod or container not in `Running` state):

```
apiVersion: v1
data:
  spec: |
    desiredVersion: "1.32"
    updateMode: Manual
  status: |
    currentVersion: "1.32"
    supportedVersions:
    - "1.30"
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    availableVersions:
    - "1.31"
    - "1.32"
    - "1.33"
    - "1.34"
    automaticVersion: "1.33"
    phase: ControlPlaneUpdating
    progress: 73%
    controlPlane:
    - name: master-1
      phase: UpToDate
      components:
        kube-apiserver: "1.32"
        kube-controller-manager: "1.32"
        kube-scheduler: "1.32"
    - name: master-2
      phase: Updating
      components:
        kube-apiserver: "1.33"
        kube-controller-manager: "1.33"
        kube-scheduler: "1.33"
    - name: master-0
      phase: Failed
      components:
        kube-apiserver: "1.32"
        kube-controller-manager: "1.32"
        kube-scheduler: "1.32"
    nodes:
      desiredCount: 6
      upToDateCount: 6
kind: ConfigMap
metadata:
```

Additional resources

- Global custom resources
- Module `control-plane-manager` documentation

Uninstallation

Deleting a cluster deployed with a cloud provider

Perform the following steps to delete a cluster deployed with a cloud provider

1. Find out the release channel set in the cluster. To do this, run the command:

   ```
   kubectl get mc deckhouse  -o jsonpath='{.spec.settings.releaseChannel}'
   ```

2. Run the Deckhouse installer:

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

where:

- `<MOUNT_OPTIONS>` — parameters for mounting files in the installer container, such as SSH access keys;
- `<DECKHOUSE_REVISION>` — the Deckhouse edition (e. g., `ee` — for the Enterprise Edition, `ce` — for the Community Edition, etc.)
- `<RELEASE_CHANNEL>` — the Deckhouse release channel in kebab-case:
  - `alpha` — for the *Alpha* release channel;
  - `beta` — for the *Beta* release channel;
  - `early-access` — for the *Early Access* release channel;
  - `stable` — for the *Stable* release channel;
  - `rock-solid` — for the *Rock Solid* release channel.

Below is an example command to run the Deckhouse CE installer container:

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

3. In the container you have started, run the following command:

```
dhctl destroy --ssh-user=<USER> \
  --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
  --yes-i-am-sane-and-i-understand-what-i-am-doing \
  --ssh-host=<MASTER_IP>
```

where:

- `<USER>` — the user of the remote machine that ran the installation. If `<USER>` needs to perform actions via `sudo`, add the `--ask-become-pass` flag;
- `<MASTER_IP>` — the IP address of the cluster's master node.

The installer will then connect to the cluster, retrieve the necessary data, and delete all the resources and objects in the cloud that were created during the DKP installation and operation.

Deleting a hybrid cluster

Follow these steps to delete a hybrid cluster consisting of the nodes that were automatically deployed in the cloud as well as the static nodes that were manually plugged in:

1. First, delete all the extra nodes (CloudStatic and Static) that were manually plugged in.
2. Find out the release channel set in the cluster. To do this, run the command:

```
kubectl get mc deckhouse  -o jsonpath='{.spec.settings.releaseChannel}'
```

3. Run the Deckhouse installer:

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

where:

- `<MOUNT_OPTIONS>` — parameters for mounting files in the installer container, such as SSH access keys;
- `<DECKHOUSE_REVISION>` — the Deckhouse edition (e. g., `ee` — for the Enterprise Edition, `ce` — for the Community Edition, etc.)
- `<RELEASE_CHANNEL>` — the Deckhouse release channel in kebab-case:
  - `alpha` — for the *Alpha* release channel;
  - `beta` — for the *Beta* release channel;
  - `early-access` — for the *Early Access* release channel;
  - `stable` — for the *Stable* release channel;
  - `rock-solid` — for the *Rock Solid* release channel.

Below is an example command to run the Deckhouse CE installer container:

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

4. In the container you have started, run the following command:

```
dhctl destroy --ssh-user=<USER> \
  --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
  --yes-i-am-sane-and-i-understand-what-i-am-doing \
  --ssh-host=<MASTER_IP>
```

where:

- `<USER>` — the user of the remote machine that ran the installation. If `<USER>` needs to perform actions via `sudo`, add the `--ask-become-pass` flag;
- `<MASTER_IP>` — the IP address of the cluster's master node;
- `<SSH_PRIVATE_KEY_FILE>`: Name of private key. For example, for a key with RSA encryption it can be `id_rsa`, and for a key with ED25519 encryption it can be `id_ed25519`.

The installer will then connect to the cluster, retrieve the necessary data, and delete all the resources and objects in the cloud that were created during the DKP installation and operation.

Deleting a static cluster

Follow the steps below to delete a cluster that has been manually installed (e.g., on bare metal):

1. Delete all the extra nodes from the cluster.
2. Find out the release channel set in the cluster. To do this, run the command:

   ```
   kubectl get mc deckhouse  -o jsonpath='{.spec.settings.releaseChannel}'
   ```

3. Run the Deckhouse installer:

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

   where:

   - `<MOUNT_OPTIONS>` — parameters for mounting files in the installer container, such as SSH access keys;
   - `<DECKHOUSE_REVISION>` — the Deckhouse edition (e.g., `ee` — for the Enterprise Edition, `ce` — for the Community Edition, etc.)
   - `<RELEASE_CHANNEL>` — the Deckhouse release channel in kebab-case:
     - `alpha` — for the *Alpha* release channel;
     - `beta` — for the *Beta* release channel;
     - `early-access` — for the *Early Access* release channel;
     - `stable` — for the *Stable* release channel;
     - `rock-solid` — for the *Rock Solid* release channel.

   Below is an example command to run the Deckhouse CE installer container:

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

4. Run the command below to delete the cluster:

   ```
   dhctl destroy --ssh-user=<USER> \
     --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> \
     --yes-i-am-sane-and-i-understand-what-i-am-doing \
     --ssh-host=<MASTER_IP>
   ```

   where:

   - `<USER>` — the user of the remote machine that ran the installation. If `<USER>` needs to perform actions via `sudo`, add the `--ask-become-pass` flag;
   - `<MASTER_IP>` — the IP address of the cluster's master node;
   - `<SSH_PRIVATE_KEY_FILE>`: Name of private key. For example, for a key with RSA encryption it can be `id_rsa`, and for a key with ED25519 encryption it can be `id_ed25519`.

The installer will then connect to the master node and delete all Deckhouse and Kubernetes cluster components on it.

Additional resources

- Module `deckhouse` parameters
- Module `node-manager` documentation

Architecture

This section describes the architecture of Deckhouse Kubernetes Platform (DKP).

The section consists of the following subsections:

- C4 model: Overview of the C4 model used to visualize the platform architecture, as well as a description of the DKP architecture at levels 1 and 2 of the C4 model.
- Modules: Description of the DKP module architecture.
- Disaster resilience: Description of the disaster resilience approaches implemented in DKP.
- Updating: Description of the DKP update mechanisms.
- Description of platform component architecture divided in the following subsystems:
  - Deckhouse subsystem
  - Kubernetes & Scheduling subsystem
  - Cluster & Infrastructure subsystem
  - IAM subsystem
  - Security subsystem
  - Network subsystem
  - Storage subsystem
  - Observability subsystem

> **i** This section does not yet cover all DKP subsystems and modules.
> Documentation for the remaining components will be added as it becomes available.

DKP architecture

DKP is a platform for managing Kubernetes clusters in various infrastructures — from isolated server environments to public clouds. The platform includes:

- A Kubernetes cluster.
- The Deckhouse controller and the modules it manages.
- Bashible, an agent running as a service on cluster nodes that executes bash scripts to manage nodes.

Modules are grouped into subsystems according to their functional purpose. The Deckhouse controller is also implemented as a module and is the only mandatory module required for the platform to function.

The DKP architecture at the subsystem and module level is described in the C4 model subsection.

Modules

A module is a set of resources and applications designed to extend DKP functionality.

Key modules:

- `deckhouse`: The Deckhouse controller.
- `control-plane-manager`: Manages cluster control plane components.
- `node-manager`: Manages cluster nodes.

> **i** The `control-plane-manager` and `node-manager` modules are not installed when DKP is installed into an existing Managed Kubernetes cluster.

Module contents:

- Helm charts
- Addon-operator hooks
- Build rules for module components (Deckhouse components)
- Other related files

DKP uses the addon-operator project to manage modules. Refer to its documentation to learn how DKP works with modules, module hooks, and module parameters.

For more information about module architecture and developing custom modules, refer to Modules.

Additional resources

- Module `control-plane-manager` documentation
- Module `deckhouse` documentation
- Module `node-manager` documentation

DKP architecture (C4 model)

The C4 model is used to visually represent a system’s architecture and explain how it is decomposed into elements. Diagrams in this model are organized according to increasing levels of scale and detail:

- **Level 1: Context diagram.** Shows the system within its external context: user roles and other systems that directly interact with it.
- **Level 2: Container diagram.** Breaks the system down into containers (major subsystems) and shows the main interaction paths between them. In the C4 model, a container (not to be confused with Docker containers) is an executable and deployable subsystem, such as an executable file.
- **Level 3: Component diagram.** Breaks each container down into components and shows interactions between components (within a single container) and with other containers. In the C4 model, a component is a software module (subsystem) within the container's codebase. This level is described in the architectural diagrams of individual services.
- **Level 4: Code diagram.** Describes the code elements that implement the components.

Due to the large number of elements at Level 2, the complete container diagram is not shown here. Instead, the following intermediate diagrams are provided after the Level 1 diagram:

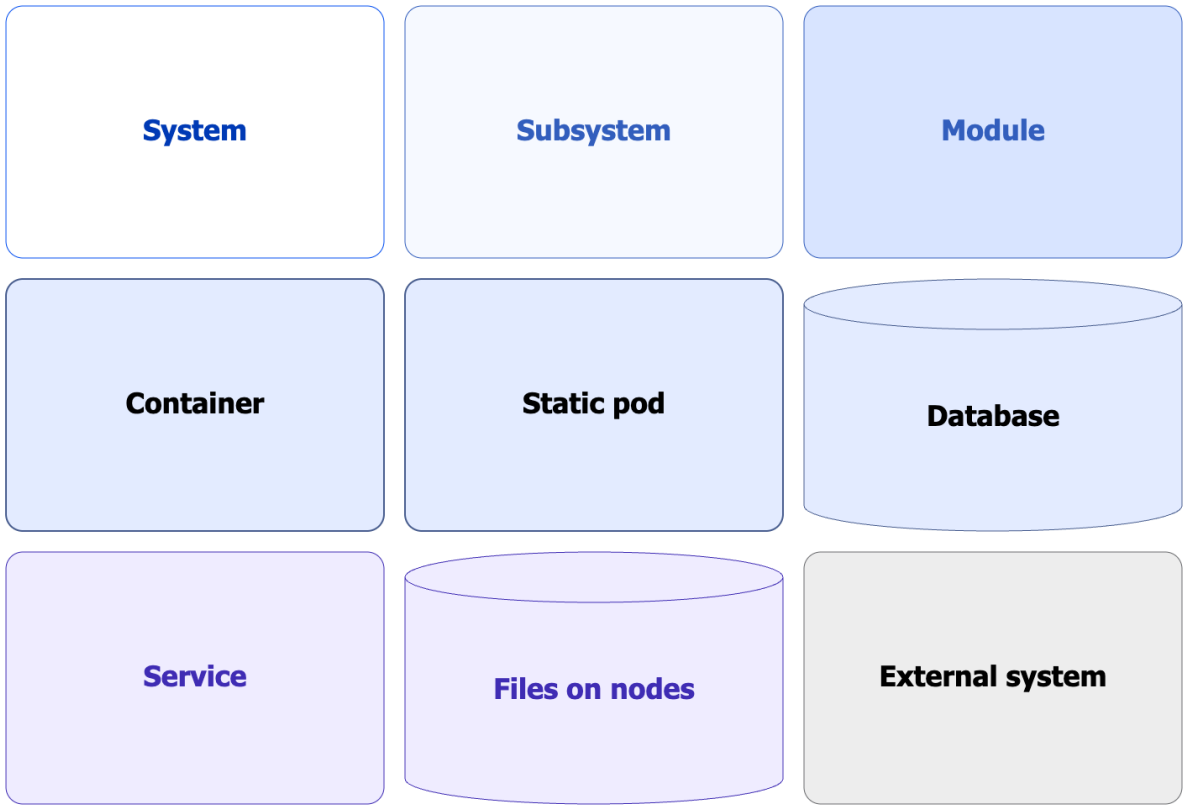
- Subsystem diagram
- Module diagram

Level 2 and higher diagrams are provided in the sections describing the architecture of individual modules.

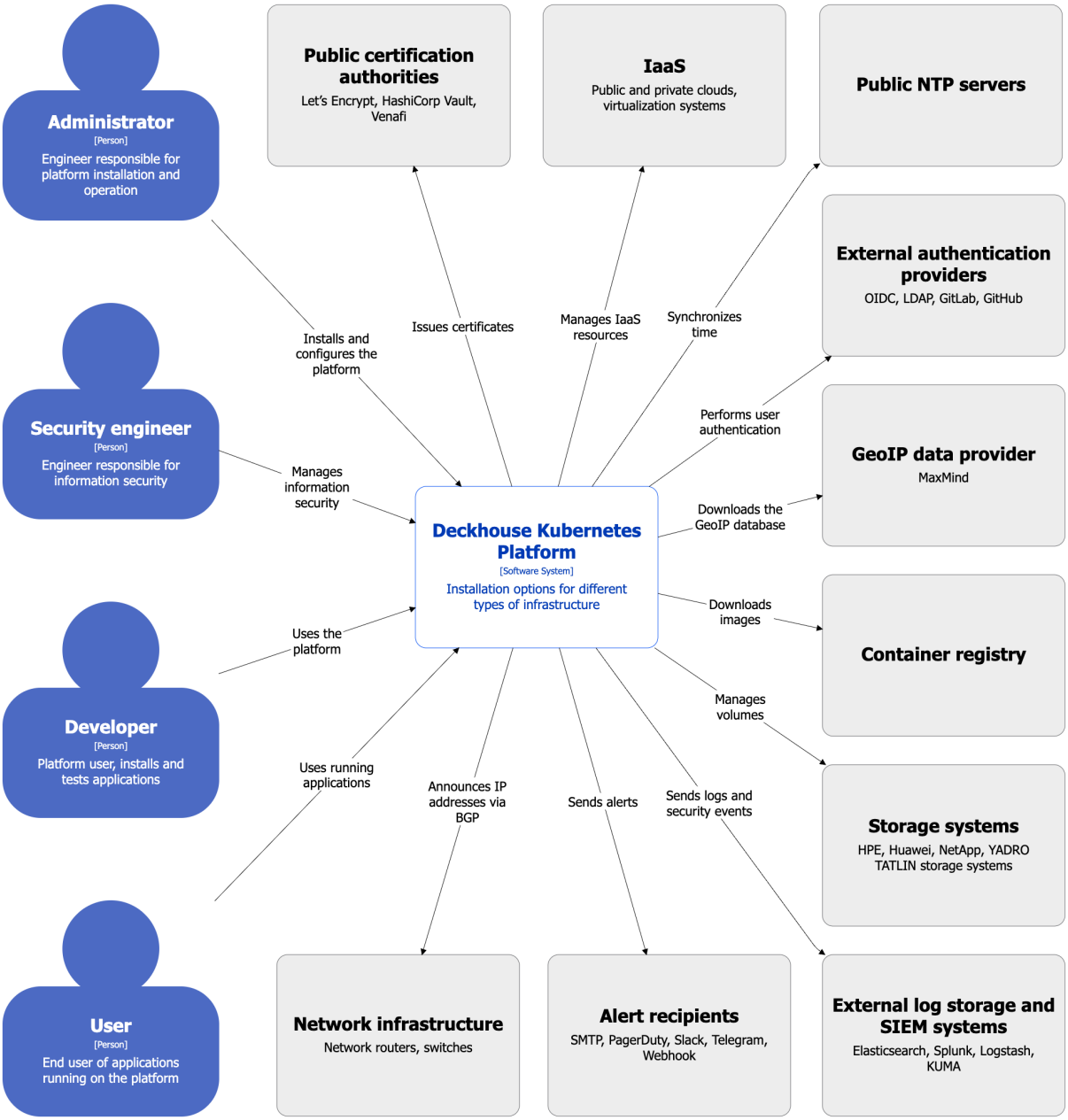
To get a general understanding of the Deckhouse Kubernetes Platform (DKP) architecture, it is sufficient to review the Level 1 diagram as well as the subsystem and module diagrams. For a detailed view of individual services, refer to the Level 2 and higher diagrams available in the relevant sections.

Legend

The following graphical symbols are used in the diagrams:



Context diagram



The following roles interact with DKP:

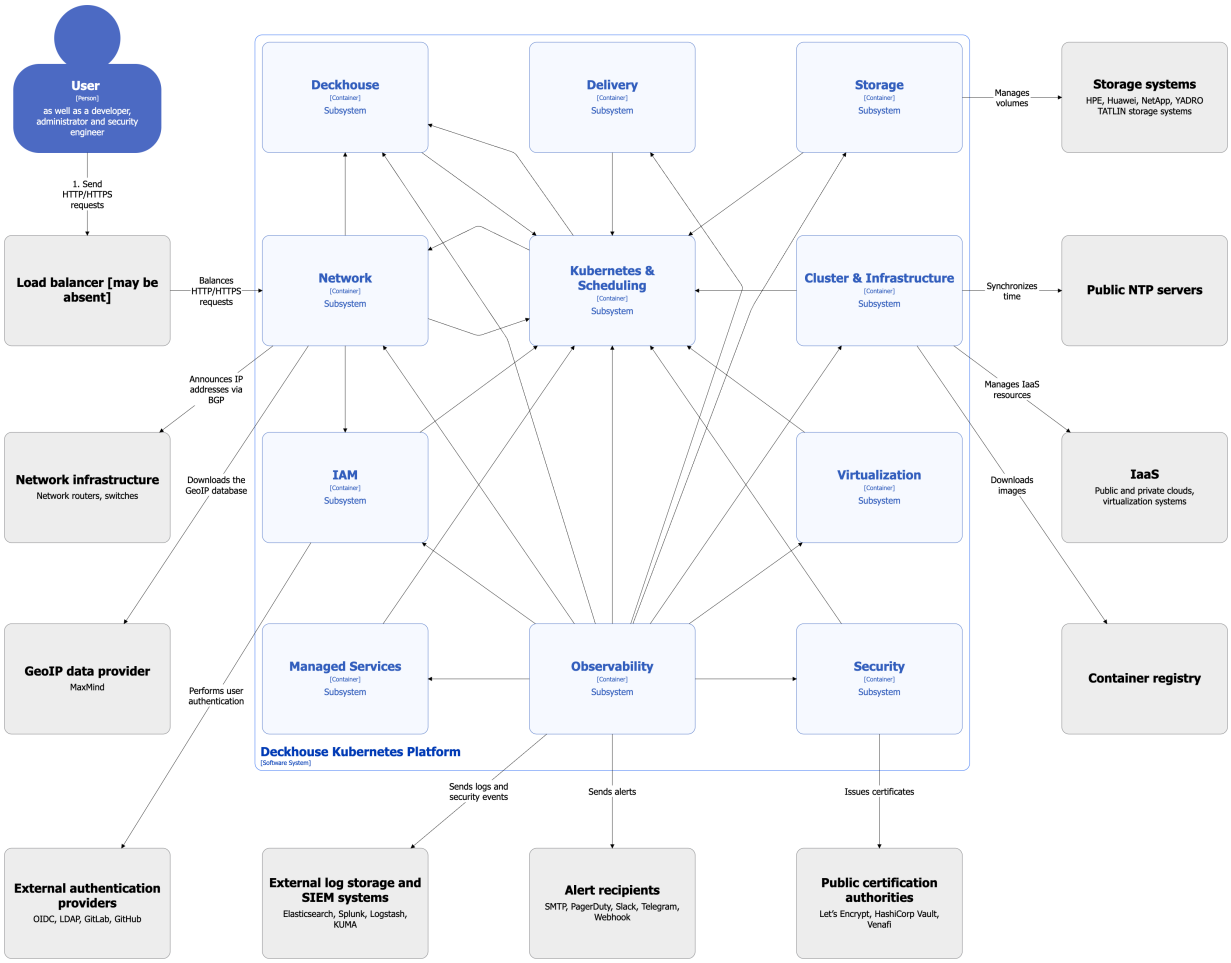
- **Administrator**: Installs and configures the platform.
- **Security Engineer**: Manages information security.
- **Developer**: Uses the platform to deploy and test applications under development.
- **User**: Uses applications running on the platform.

Interactions with external services are described in the diagrams of subsequent levels.

Subsystem diagram

The diagram below shows the DKP subsystems.

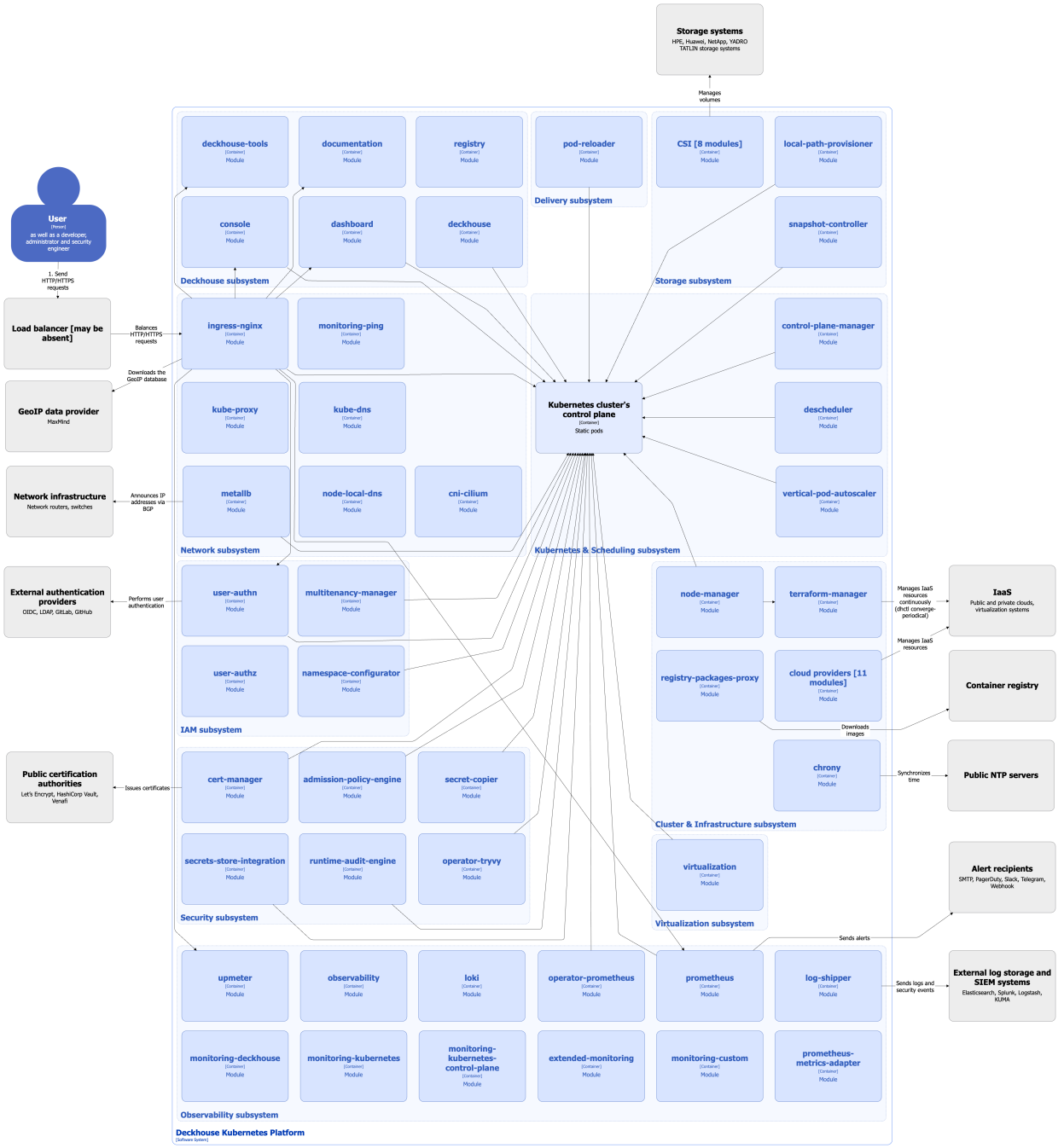
> **i** Relationships between subsystems are shown in a generalized form and without detailed interactions.



Module diagram

The diagram below shows all modules included in the Default module bundle, as well as the most important additional modules. Modules are grouped by subsystem.

> **i** Relationships between modules are shown in a generalized form and without detailed interactions.



Additional resources

- The C4 model for visualising software architecture

Deckhouse Kubernetes Platform module development

A module is a set of resources and applications designed to extend the functionality of the Deckhouse Kubernetes Platform.

This section is dedicated to the architecture of DKP modules. It will help you understand how modules are structured, how to develop and debug them, and how they interact with other components of the platform.

Deckhouse Kubernetes Platform (DKP) can work with the following types of modules:

- Built-in modules. These are included in DKP. Their release cycle is tied to the DKP release cycle.
- Modules from module source. The release cycle for these modules is not tied to the DKP release cycle.

Creating a module consists of the following stages:

- **Development**: creating module code and its structure in the Git repository. The **Module structure** section outlines which components there are and in which directories they are located.
- **Building**: creating a module artifact and pushing it to the container registry. The **Building and publishing** section describes where images are stored in the registry and at what paths they are available.
- **Running in a cluster**: delivering the module to a cluster managed by the DKP. The **Running in a cluster** section describes how to activate the module, configure its parameters, and test its functionality

(including handling CRDs and troubleshooting).
- **Dependencies**: configuring module dependencies, including DKP versions, Kubernetes, and other critical components. This stage is covered in the **Module dependencies** section.

Requirements

You will need the following tools to develop DKP modules:

- git — version control system;
- sed — stream editor;
- yq — CLI tool for processing data in JSON, YAML, and XML formats;
- jq — CLI tool for processing data in JSON, YAML, and XML formats;
- werf — (optional) CLI tool for building images. You will need it for building module artifacts locally;
- crane — (optional) CLI tool for working with the container registry. You may need it when debugging.

The container registry where module artifacts will be stored must support a nested repository structure. A registry such as Docker Registry v2 or Harbor is a good choice.

Before you start

To get an idea of how DKP modules work, check out addon-operator and shell-operator.

- Review the operator documentation on the concept of hooks, e.g., what a hook configuration is and what functions it provides. The configuration is used to configure the data that will be available from the hook.
- Check out bindings. Bindings are events that trigger the hook. They are specified in the hook configuration. A hook can be triggered not only by Kubernetes events, but also, e.g., on a schedule or before a module is started.

  > **i** The Hook allows you to keep values in memory and use them later when rendering Helm templates. We recommend reading the Hooks and Helm values section to learn more about this feature as well as the module's operating cycle.

- Explore the concept of snapshots. With snapshots, you can implement a reconciliation loop approach that is more efficient than event subscription.

  > **i** This is how DKP implements support for all existing backend module hooks.

- Additionally, hooks can be used instead of the Prometheus exporter. Hooks can provide metrics that DKP will export. See metrics.

Got a question?

The Development and debugging section contains information about what tools can be used in module development and approaches to troubleshooting module errors.

Join the community, where you'll be sure to have your questions answered.

Module structure

The source code of the module and its assembly rules must be located in a directory with a certain structure. The best analog is a Helm chart.

Not all module folders and files are required. In short, you can use the following guidelines:

- Create a module.yaml file to describe the module metadata.
- In the templates folder, place the Helm templates that will be used in the cluster.

  If you want the objects created by the module to change their behavior depending on any module parameters, define the necessary parameters in the specification and use them in the templates.
- In the images folder, place instructions for building container images used by the module.

  If the templates only use addresses of *external* images, you don't need to create a folder.
- If the module should respond to events or interact with the Kubernetes API, you need to create hooks that should be placed in the hooks folder.
- Place the module documentation in the docs folder. If the module documentation is missing, the module will not appear in the list of modules in the documentation web interface in the cluster.
- If the module has auxiliary Helm charts, place them in the charts folder.
- If the module needs to create custom resources (CRD), place their specifications in the crds folder.

There is a repository containing the sample module template. We recommend you start your module development with it.

Further down this page you will find a more detailed description of the module directory and file structure.

▾ Example module folder structure...

Example module folder structure containing build and publish rules using GitHub Actions:

```
 my-module/
├──  .github/
│   ├──  workflows/
│   │   ├──  build_dev.yaml
│   │   ├──  build_prod.yaml
│   │   ├──  checks.yaml
│   │   ├──  deploy_dev.yaml
│   │   └──  deploy_prod.yaml
├──  .werf/
│   ├──  workflows/
│   │   ├──  base-images.yaml
│   │   ├──  bundle.yaml
│   │   ├──  images.yaml
│   │   ├──  images-digest.yaml
│   │   ├──  batch-go.yaml
│   │   └──  release.yaml
├──  charts/
│   └──  helm_lib/
├──  crds/
│   ├──  crd1.yaml
│   ├──  doc-ru-crd1.yaml
│   ├──  crd2.yaml
│   └──  doc-ru-crd2.yaml
├──  docs/
│   ├──  README.md
│   ├──  README.ru.md
│   ├──  EXAMPLES.md
│   ├──  EXAMPLES.ru.md
│   ├──  CONFIGURATION.md
│   ├──  CONFIGURATION.ru.md
│   ├──  CR.md
│   ├──  CR.ru.md
│   ├──  FAQ.md
│   ├──  FAQ.ru.md
│   ├──  ADVANCED_USAGE.md
│   └──  ADVANCED_USAGE.ru.md
├──  hooks/
│   ├──  batch/
│   │   ├──  my-hooks/
│   │   │   ├──  my-hook1.go
│   │   │   └──  my-hook2.go
│   │   ├──  go.mod
│   │   ├──  go.sum
│   │   ├──  main.go
├──  images/
│   ├──  nginx
│   │   └──  Dockerfile
│   └──  backend
│       └──  werf.inc.yaml
├──  lib/
│   └──  python/
│       └──  requirements.txt
├──  openapi/
│   ├──  conversions
│   │   ├──  testdata
│   │   │   ├──  v1-1.yaml
│   │   │   └──  v2-1.yaml
│   │   ├──  conversions_test.go
│   │   └──  v2.yaml
│   ├──  config-values.yaml
│   ├──  doc-ru-config-values.yaml
│   └──  values.yaml
├──  templates/
│   ├──  a.yaml
│   └──  b.yaml
├──  .helmignore
├──  Chart.yaml
├──  module.yaml
├──  werf.yaml
└──  werf-giterminism.yaml
```

charts

The `/charts` directory contains Helm helper charts used when rendering templates.

Deckhouse Kubernetes Platform (DKP) has its own library for working with templates called lib-helm. You can read about the library's features in the lib-helm repository. To add the library to the module, download the tgz-archive with the appropriate release and move it to the `/charts` directory of the module.

crds

This directory contains *CustomResourceDefinitions* (CRDs) used by the module components. CRDs are updated every time the module is started, if there are updates.

Nested directories in the `/crds` folder are ignored.

To render CRDs from the `/crds` directory in the site documentation or documentation module in the cluster, follow these steps:

- Create a translation file with a structure identical to the original resource file:
  - in it, keep only the `description` parameters containing the translation text;
  - use the `doc-ru-` prefix in the name: e.g., `/crds/doc-ru-crd.yaml` for `/crds/crd.yaml`.
- Create `/docs/CR.md` and `/docs/CR.ru.md` files.

docs

> **i** The module lifecycle is specified in module.yaml. The availability of the module in the editions of the Deckhouse Kubernetes Platform is not determined by the module developer.

The `/docs` directory contains the module documentation. The following nested directories aren't involved in the documentation building process:

- `internal`
- `internals`
- `development`
- `dev`

The following files are required for building the documentation:

- `README.md` and `README.ru.md` — this file (and its localized version) describes what the module is for, what problem it solves and outlines the general architectural principles.

  The (front matter) file metadata as a YAML structure must present in all language versions of the file. You can use the following parameters in the metadata:

  - `title` — **(recommended)** The title of the module description page, for example, "Deckhouse web admin console". It is also used in navigation if `linkTitle` parameter is not specified.
  - `menuTitle` — **(recommended)** The name of the module to show in the menu on the left sidebar of the page, e.g., "Deckhouse Admin". If not set, the name of the directory or repository is used, e.g. `deckhouse-admin`.
  - `linkTitle` — **(optional)** Alternative title for navigation if, for example, the `title` is very long. If not set, the `title` parameter is used.
  - `description` — **(recommended)** A short unique description of the page content (up to 150 characters). It should not repeat the `title`. Goes on with the meaning of the title and reveals it in more detail. It is used during generation of preview links and indexing by search engines, e.g., "The module allows you to fully manage your Kubernetes cluster through a web interface with only mouse skills."

  ▼ **Metadata example...**

  ```
  ---
  title: "Deckhouse administrator web console"
  menuTitle: "Deckhouse Admin"
  description: "The module allows you to fully manage your Kubernetes cluster through a web interface with
  only mouse skills."
  ---
  ```

The following files aren't required but have a predetermined name in the sidebar menu and the page title:

- `EXAMPLES.md` and `EXAMPLES.ru.md` — this file (and its localized version) contains examples of module configuration with description.

  The (front matter) file metadata as a YAML structure must present in all language versions of the file. You can use the following parameters in the metadata:

  - `title` – **(recommended)** The title of the page, e.g., `Examples`. It is also used in navigation if there is no `linkTitle`.
  - `description` – **(recommended)** A short unique description of the page content (up to 150 characters). It should not repeat the `title`. Goes on with the meaning of the title and reveals it in more detail. It is used during generation of preview links and indexing by search engines, e.g., "Examples of storing secrets in a neural network and automatically substituting them into thoughts when communicating."
  - `linkTitle` – **(optional)** Alternative title for navigation if, for example, the `title` is very long. If not set, the `title` parameter is used.

  ▼ **Metadata example...**

```
---
title: "Examples"
description: "Examples of storing secrets in a neural network with automatic substitution into thoughts
when communicating."
---
```

- `FAQ.md` and `FAQ.ru.md` — this file (and its localized version) contains frequently asked questions related to module operation, e.g., “What scenario should I choose: A or B?”.

  The (front matter) file metadata as a YAML structure must present in all language versions of the file. You can use the following parameters in the metadata:

  - `title` – **(recommended)** The title of the page.
  - `description` – **(recommended)** A short unique description of the page content (up to 150 characters).
  - `linkTitle` – **(optional)** Alternative title for navigation if, for example, the `title` is very long. If not set, the `title` parameter is used.

  ▼ **Metadata example...**

  ```
  ---
  title: "FAQs"
  description: "Frequently asked questions."
  ---
  ```

- `ADVANCED_USAGE.md` and `ADVANCED_USAGE.ru.md` — this file (and its localized version) contains expanded instructions on using and debugging the module.

  The (front matter) file metadata as a YAML structure must present in all language versions of the file. You can use the following parameters in the metadata:

  - `title` – **(recommended)** The title of the page.
  - `description` – **(recommended)** A short unique description of the page content (up to 150 characters).
  - `linkTitle` – **(optional)** Alternative title for navigation if, for example, the `title` is very long. If not set, the `title` parameter is used.

  ▼ **Metadata example...**

  ```
  ---
  title: "Module debugging"
  description: "This section covers all the steps for debugging the module."
  ---
  ```

- `CR.md` and `CR.ru.md` — the files for generating resources from the `/crds/` directory. Add these files if such a generation is required.

  ▼ **Metadata example...**

  ```
  ---
  title: "Custom resources"
  ---
  ```

- `CONFIGURATION.md` and `CONFIGURATION.ru.md` — this file (and its localized version) is used to generate OpenAPI specifications from `/openapi/config-values.yaml` and `/openapi/doc-<LANG>-config-values.yaml`. Add these files if such a generation is required.

  ▼ **Metadata example...**

  ```
  ---
  title: "Module settings"
  ---
  ```

All images, PDF files and other media files should be stored in the `/docs` directory or its subdirectories (e.g, `/docs/images/`). All links to files should be relative.

You need a file with the appropriate suffix for each language, for example `image1.jpg` and `image1.ru.jpg`. Use the following links:

- `[image1](image1.jpg)` in an English-language document;
- `[image1](image1.ru.jpg)` in a Russian-language document.

hooks

The `/hooks/batch` directory contains the module’s hooks. A hook is an executable file executed in response to an event. Hooks are also used by the module for dynamic interaction with Kubernetes API. For example, they can be used to handle events related to the creation or deletion of objects in a cluster.

Get to know the concept of hooks before you start developing your own hook. You can use the Go library by the Deckhouse team to speed up the development of hooks.

Hook requirements:

- When run with the `hook config` arguments, it should output its hook configuration in JSON format.
- When run with the `hook list` arguments, it should output a run of all hooks with their sequence number.
- When run with the `hook run 0` arguments, the logic of the hook with number 0 should be executed.

The hook files must be executable. Add the appropriate permissions using the `chmod +x <path to the hook file>` command.

You can find Go hook examples in the module template repository. Go hook examples can also be found in the SDK.

images

The `/images` directory contains instructions for building module container images. The first level contains directories for files used to create the container image, the second level contains the building context.

There are two ways to define a container image:

1. Dockerfile — this file contains commands for building images. To build an application from source code, copy it next to the Dockerfile and include it in the image using the `COPY` command.
2. The `werf.inc.yaml` file, which is the same as the image definition section in `werf.yaml`.

The image name matches the directory name for this module, written in *camelCase* notation starting with a small letter. For example, the directory `/images/echo-server` corresponds to the image name `echoServer`.

The built images have content-based tags that can be used when building other images. To use content-based image tags, enable the lib-helm library. You can also use other features of the helm_lib library of Deckhouse Kubernetes Platform.

Below is an example of using a content-based image tag in a Helm chart:

```
image: {{ include "helm_lib_module_image" (list . "<image name>") }}
```

openapi

conversions

The `/openapi/conversions` directory contains module parameter conversion files and their tests.

Module parameter conversions allow you to convert the OpenAPI specification of module parameters from one version to another. Conversions may be necessary when a parameter is renamed or moved to a different location in a new version of the OpenAPI specification.

Each conversion can only be performed between two consecutive versions (e.g., from the first to the second one). There can be several conversions, and the chain of conversions must sequentially cover all versions of the parameter specification with no “gaps”.

The conversion file is a YAML file named `v<N>.yaml` or `v<N>.yml`, where `<N>` is the conversion version. It has the following structure:

```
# Version number of the module parameter specification to which the data is converted
during the conversion.
version: N
# A set of jq expressions used during conversion in the cluster for the automatic
transformation
# of the previous version module parameters.
conversions: []
# Actions (in two languages) that need to be taken to convert the data
# from the previous version of the module parameter specification.
description:
  ru: ""
  en: ""
```

The description of actions specified in the description section of the conversion file must be clear and include information on which parameters and in what order need to be changed in the module’s parameter specification in order to switch to the new version.

Below is an example of a module parameter conversion file `v2.yaml`, where in version 2 the `.auth.password` parameter has been removed:

```
version: 2
conversions:
  - del(.auth.password) | if .auth == {} then del(.auth) end
description:
  ru: "Remove `.auth.password`, then `auth` if empty."
  en: "Remove `.auth.password`, then `auth` if empty."
```

Conversion tests

You can use the `conversion.TestConvert` function to write conversion tests. It receives the following parameters:

- Path to the source configuration file (i.e., the version before the conversion);
- Path to the resulting configuration file (i.e., the version after the conversion).

An example of a conversion test.

config-values.yaml

This file is required to validate the module parameters that the user can configure via ModuleConfig.

To render the schema in the documentation on the site or in the documentation module in the cluster, create:

- The `doc-ru-config-values.yaml` file with a structure similar to that of the `config-values.yaml` file. Keep only the translated description parameters in the `doc-ru-config-values.yaml` file;
- The `/docs/CONFIGURATION.md` and `/docs/CONFIGURATION.ru.md` files to enable rendering of data from the `/openapi/config-values.yaml` and `/openapi/doc-ru-config-values.yaml` files.

An example of a `/openapi/config-values.yaml` schema with a single configurable `nodeSelector` parameter:

```
type: object
properties:
  nodeSelector:
    type: object
    additionalProperties:
      type: string
    description: |
      The same as the Pods' `spec.nodeSelector` parameter in Kubernetes.

      If the parameter is omitted or `false`, `nodeSelector` will be determined
      automatically.</code>
```

An example of the `/openapi/doc-ru-config-values.yaml` file:

```
properties:
  nodeSelector:
    description: |
      English description. Markdown markup.</code>
```

Validations with x-deckhouse-validations

When developing a module for the Deckhouse Kubernetes Platform, you can use the OpenAPI extension `x-deckhouse-validations` to describe complex validation rules for module parameters using Common Expression Language (`CEL`).

When using these validation rules, keep the following features in mind:

- Validations can be placed at the root level or inside any property (including inside objects, arrays, and additionalProperties).
- All parameters at the current level are available in expressions via the `self` variable.
- The previous value at the same level is available in expressions via the `oldSelf` variable. Rules that reference `oldSelf` are treated as transition rules and are evaluated only on updates (when a previous value exists at this level); on the initial create or for newly added subtrees they are silently skipped. This is the same model as `x-kubernetes-validations`.
- Validation works recursively: all nested objects, arrays, and maps can also contain their own `x-deckhouse-validations`.
- Supported types: scalars, arrays, objects, and maps (`additionalProperties`).
- If there are multiple validation errors, the user will see all messages from the corresponding rules.

Examples of rules

Below are examples of complex validation rules described in CEL:

- Checking whether the parameter value falls within the range:

```
type: object
properties:
  replicas:
    type: integer
  minReplicas:
    type: integer
  maxReplicas:
    type: integer
x-deckhouse-validations:
  - expression: "self.minReplicas <= self.replicas && self.replicas <=
self.maxReplicas"
    message: "replicas must be between minReplicas and maxReplicas"
```

- Checking for the presence of a key:

```
- expression: "'Available' in self.stateCounts"
  message: "The key 'Available' must be present"
```

- Checking that exactly one of two lists is non-empty:

```
- expression: "(self.list1.size() == 0) != (self.list2.size() == 0)"
  message: "Exactly one of the lists must be non-empty"
```

- Checking a value by regular expression:

```
- expression: "self.details.all(key, self.details[key].matches('^[a-zA-Z]*$'))"
  message: "All values must contain only letters"
```

- Marking a field as immutable using the `oldSelf` variable (transition rule):

```
type: object
properties:
  clusterDomain:
    type: string
x-deckhouse-validations:
  - expression: "self.clusterDomain == oldSelf.clusterDomain"
    message: "clusterDomain is immutable"
```

  The rule is only evaluated on updates: on the initial create (when there is no previous value) it is automatically skipped, just like a transition rule in `x-kubernetes-validations`.

- Allowing a list to only grow (each existing item must be preserved on update):

```
- expression: "oldSelf.items.all(x, x in self.items)"
  message: "items can only be appended to, not removed"
```

Scalar and array value validation

Validation of scalar values and arrays has the following features:

- If the property is a scalar (e.g., number or string), then in the `CEL` expression the `self` variable will be that value.
- If the property is an array, then `self` will be an array, and you can use methods like `.size()`, `.all()`, `.exists()`, etc.

Example for an array:

```
type: object
properties:
  items:
    type: array
    items:
      type: string
    x-deckhouse-validations:
      - expression: "self.size() > 0"
        message: "The items list must not be empty"
```

Validation of additionalProperties (map)

For objects with additionalProperties (map), you can validate keys and values using methods like `.all(key, ...)`, `.exists(key, ...)`, etc.

Example:

```
type: object
properties:
  mymap:
    type: object
    additionalProperties:
      type: integer
    x-deckhouse-validations:
      - expression: "self.all(key, self[key] > 0)"
        message: "All values in mymap must be greater than 0"
```

values.yaml

This file is required for validating the source data when rendering templates without using extra Helm chart functions. Its closest analogs are Helm's schema files.

You can automatically add parameter validation from `config-values.yaml` to `values.yaml`. In this case, the basic `values.yaml` looks as follows:

```
x-extend:
  schema: config-values.yaml
type: object
properties:
  internal:
    type: object
    default: {}
```

templates

The `/templates` directory contains Helm templates.

- Use the path `.Values.<moduleName>` to access module settings in templates, and `.Values.global` for global settings. The module name is converted to `camelCase` notation.
- To facilitate working with templates, use lib-helm, which is a set of extra functions that make it easier to work with global and module values.
- Accesses to the registry from the ModuleSource resource are available at the `.Values.<moduleName>.registry.dockercfg` path.
- To use these functions to pull image pools in controllers, create a secret and add it to the corresponding parameter: `"imagePullSecrets": [{"name":"registry-creds"}]`.

  ```
  apiVersion: v1
  kind: Secret
  metadata:
    name: registry-creds
  type: kubernetes.io/dockerconfigjson
  data:
    .dockerconfigjson: {{ .Values.<moduleName>.registry.dockercfg }}
  ```

A module can have parameters with which it can alter its behavior. Module parameters and their validation

scheme are described in OpenAPI-schemes in `/openapi` directory.

The settings are stored in two files: `config-values.yaml` and `values.yaml`.

You can find an example of an OpenAPI schema in module template.

.helmignore

`.helmignore` allows you to exclude files from the Helm release. In case of DKP modules, directories `/crds`, `/images`, `/hooks`, `/openapi` must be added to `.helmignore` to avoid exceeding 1 Mb limit of Helm release size.

Chart.yaml

This is a file for a chart, similar to `Chart.yaml` in Helm. It must contain at least a `name` parameter with the module name and a `version` parameter with the version. You don’t need to create this file, Deckhouse will create it automatically.

An example:

```
name: echoserver
version: 0.0.1
dependencies:
- name: deckhouse_lib_helm
  version: 1.38.0
  repository: https://deckhouse.github.io/lib-helm
```

module.yaml

The `module.yaml` file in the root of the module folder contains the module’s metadata.

The file might not be present, but it is recommended to fill it in. Most of the metadata will be available in the Module object in the cluster. The Module object will be created automatically after the module source (resource ModuleSource) is configured and synchronization is successful.

Parameters that can be used in `module.yaml`:

Parameter	Type	Description
`namespace`	String	Namespace where the module components will be deployed
`subsystems`	Array of strings	List of subsystems the module belongs to
`accessibility`	Object	Module accessibility settings
`accessibility.editions`	Object	Module operation settings in DKP editions
`accessibility.editions.available`	Boolean	Defines whether the module is available in a DKP edition
`accessibility.editions.enabledInBundles`	Array of strings	List of module bundles in which the module should be enabled by default
`descriptions`	Object	Arbitrary text description of the module’s purpose
`descriptions.en`	String	Description in English
`descriptions.ru`	String	Description in Russian
`disable`	Object	Parameters related to the behavior when disabling a module
`disable.confirmation`	Boolean	Requires confirmation when disabling the module
`disable.message`	String	Message explaining what will happen when the module is disabled

If confirmation is required to disable a module (`confirmation` is set to `true`), then disabling the module is only possible if the corresponding ModuleConfig object has the annotation `modules.deckhouse.io/allow-disabling=true`. If this annotation is not present, Deckhouse will block the disabling attempt and display the warning message from the `message` field.

Other `module.yaml` parameters:

Parameter	Type	Description
`name`	String, mandatory parameter	Name of the module in Kebab Case. For example, `echo-server`
`exclusiveGroup`	String	If several modules share the same value of this parameter, only one of them can be active in the system at any given time. This prevents conflicts between modules performing similar or incompatible functions
`requirements`	Object	Module dependencies — a set of conditions that must be met for Deckhouse Kubernetes Platform (DKP) to run the module
`requirements.deckhouse`	String	Dependency on the Deckhouse Kubernetes Platform version that the module is compatible with
`requirements.kubernetes`	String	Dependency on the Kubernetes version that the module is compatible with
`requirements.modules`	Object	Dependency on the version of other modules
`stage`	String	Module lifecycle stage. Possible values: `Experimental`, `Preview`, `General Availability`, `Deprecated`. If `stage` is set to `Experimental`, the module cannot be enabled by default. To allow the use of such modules, set the corresponding parameter to `true`
`tags`	Array of strings	Additional module tags. Tags are converted to labels of the Module object using the template `module.deckhouse.io/<TAG>=""`. For example, if you specify two tags `test` and `example`, the Module object will get the labels `module.deckhouse.io/test=""` and `module.deckhouse.io/example=""`
`weight`	Number	Module weight. Affects the module startup order: modules with a lower `weight` start earlier. Default: `900`. The startup order is also affected by module dependencies
`critical`	Boolean	Marks the module as critical for the initial cluster bootstrap. Such modules (if used in the cluster) start during the initial cluster boot, before the cluster is considered fully ready. Other modules (non-critical) start only after the cluster is fully ready. Default: `false`

Example of metadata description for the `hello-world` module:

```
name: hello-world
tags: ["test", "myTag"]
weight: 960
stage: "Experimental"
namespace: "test"
exclusiveGroup: "group"
subsystems:
  - test
  - test1
accessibility:
  editions:
    ee:
      available: true
      enabledInBundles:
        - Default
descriptions:
  en: "The module to say hello to the world."
  ru: "Модуль, который приветствует мир."
requirements:
  deckhouse: ">= 1.61"
  kubernetes: ">= 1.27"
disable:
  confirmation: true
  message: "Disabling this module will delete all resources created by the module."
```

Configuring module accessibility in DKP editions

The `accessibility` parameter lets you define the DKP editions and module bundles in which a module is available, and whether it should be enabled by default.

```
accessibility:
  editions:
    _default:
      available: true
      enabledInBundles:
        - Default
        - Managed
    ce:
      available: false
    ee:
      available: true
      enabledInBundles:
        - Managed
```

Parameter description:

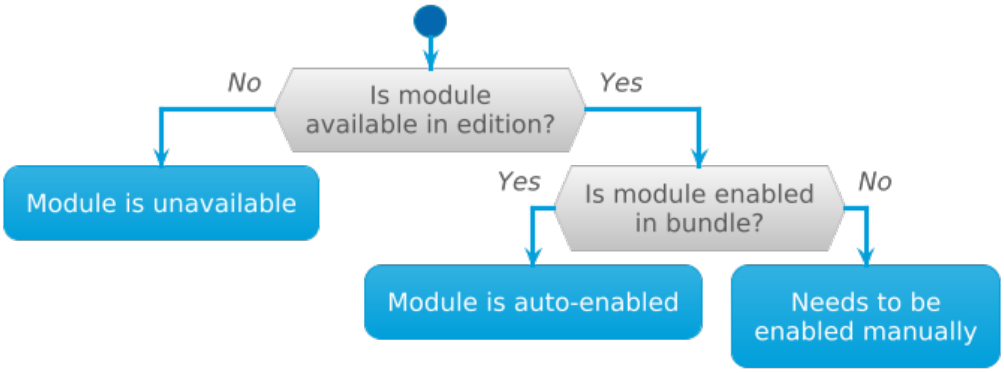
- `accessibility`: *Object.* Root block for configuring module accessibility.
- `editions`: *Object.* Set of keys representing edition names. Each edition can have its own accessibility settings.
- `_default`: *Object.* Default configuration used when no settings are defined for a specific edition.
- `available`: *Boolean.* Defines whether the module is available in the specified edition.
- `enabledInBundles`: *Array of strings.* Module bundles in which the module is enabled by default.
  Supported module bundles (the full contents of each bundle are listed on this page):
  - `Default`: Recommended set of modules for running a cluster. Includes monitoring, authorization control, networking, and other essential components.
  - `Managed`: Set of modules for clusters managed by cloud providers (for example, Google Kubernetes Engine).
  - `Minimal`: Minimal set that includes only the current module.

    > ⚠ Note that basic modules (such as the CNI module) are not included in this set. Without basic modules, DKP can only operate in an already deployed cluster. The list of modules that must be enabled manually during installation is given in “Things to keep in mind when working with the Minimal module set”.

- Sections containing edition names. Define module behavior in specified editions. Possible values: `be`, `ce`, `ee`, `se`, `se-plus`.

Module accessibility logic

The following diagram illustrates the logic for determining module accessibility and whether it is enabled by default:



Configuration examples

In the following example, the module will be unavailable in all editions except DKP Enterprise Edition. In DKP Enterprise Edition, the module will be enabled by default in the `Managed` bundle.

```
accessibility:
  editions:
    _default:
      available: false
    ee:
      available: true
      enabledInBundles:
        - Managed
```

In the next example, the module will be available in all DKP editions. In the `Managed` and `Default` bundles, the module will be enabled by default.

```
accessibility:
  editions:
    _default:
      available: true
      enabledInBundles:
        - Managed
        - Default
```

In the following example, the module will be available in all DKP editions. It will be enabled in the `Default` and `Managed` bundles for all editions, except for DKP Basic Edition and DKP Community Edition.

```
accessibility:
  editions:
    _default:
      available: true
      enabledInBundles:
        - Default
        - Managed
    be:
      available: false
    ce:
      available: false
```

Additional resources

- Global custom resources
- Global parameters
- Module `deckhouse` parameters

Build and publish

Deckhouse Kubernetes Platform (DKP) uses container registry to pull a module and update it. The container

registry stores module artifacts. Module artifacts are created when a module is built and can then be uploaded (published) to the registry.

Types of module artifacts

Module builds create three types of artifacts that are pushed to the container registry:

- **Images of application containers**. The build rules and source code for these images are stored in the images subdirectory of the *application name* directory. The built images are then specified in the templates and run in the cluster. Images are tagged with the content-based tags. Note that the lib-helm library must be enabled to use them in the templates.
- **Module image**. The module assembly rules are in the `werf.yaml` file in the module directory. The semantic versioning is used as image tags.
- **Release**. Module version artifact. Based on the release data, DKP decides whether to update a module in the cluster. Releases have two types of tags: a semantic versioning tag (just like with the module image) and a tag that matches the release channel (e.g., `alpha`, `beta`, etc.). In the module template there is an example of workfklow for GitHub Actions, where the release is created automatically during building.

Building module artifacts and publishing them to the container registry

We suggest using the pre-compiled GitHub Actions to build module artifacts and upload them to the container registry as part of the CI/CD process.

The module template repository provides an example module. It contains a basic workflow for GitHub Actions and uses GitHub Packages (ghcr.io) as the container registry. The example workflow follows the logic outlined below:

- Build module artifacts when changes are made to the PR and when changes are merged into the main branch.
- Build module artifacts from tags using the production container registry.
- Publish module to GitHub Packages container registry to the selected release channel from a tag.

Module artifacts will be uploaded to `ghcr.io/<OWNER>/modules/`, which will serve as module source.

For the module workflow to run smoothly, adjust the properties of your project on GitHub as follows:

- Open the *Settings -> Actions -> General* page.
- Enable the *Read and write permissions* parameter in the *Workflow permissions* section.

You can also modify the workflow, use your own container registry, and design a more complex build and publish process (e.g., use separate container registries for development and production).

⚠ When developing **multiple modules** and publishing them to GitHub Packages, you should use your account's Personal Access Token (PAT).

Do not use `GITHUB_TOKEN` in GitHub Workflows. This way, you can avoid access rights issues when uploading images, as the final release images are stored at `ghcr.io/<OWNER>/modules/` — the address that belongs to the first repository created.

Here is how you can adapt a module template to use a PAT:

- On the *Settings -> Secrets and variables -> Actions* page create a Secret named `TOKEN` containing a PAT.
- Replace the `GITHUB_TOKEN` variable with `TOKEN` in `.github/workflows/`:

```
cd <REPO>
sed -i -e 's/GITHUB_TOKEN/TOKEN/g' $(find .github/workflows/ -type f)
```

i You can build module artifacts locally using werf (this may come in handy, for example, when debugging).

You can also set up a custom module artifact build and publish processes for your CI/CD system similar to the workflow for GitHub Actions provided in the module template. However, this may require an in-depth understanding of the module build and publish processes. Please contact the community should you have any questions or issues.

Here is a general scenario for using the workflow provided in the module template:

1. Publish the changes to the module code in the project branch on GitHub. This will trigger the module artifacts to be built and published to the container registry.
2. Create a new module release or add a tag in the semantic versioning format to the target commit.
3. Go to the *Actions* section of the module repository on GitHub and select *Deploy* in the workflow list on the

left.
4. On the right side of the page, click on the *Run workflow* drop-down list, select the desired release channel, and enter the target tag in the tag input box. Click the *Run workflow* button.
5. Once the workflow is successfully executed, a new version of the module will be added to the container registry. The published version of the module can then be used in the cluster.

Additional resources

- Global custom resources

How to run and verify a module in the DKP cluster

This section describes the process of running a module in a Deckhouse Kubernetes Platform (DKP) cluster, as well as connecting Deckhouse Module Tools for setting up validation and metrics collection.

Run the module in the DKP cluster

Follow these steps to run the module in a cluster:

- Define ModuleSource (the ModuleSource resource).
- *(optional)* Define the module update policy (the ModuleUpdatePolicy resource).
- Enable the module in the cluster (the ModuleConfig resource).

Module source

Deckhouse Kubernetes Platform (DKP) can work with the following types of modules:

- Built-in modules. These are included in DKP. Their release cycle is tied to the DKP release cycle.
- Modules from module source. The release cycle for these modules is not tied to the DKP release cycle.

Create a ModuleSource resource to set the source to fetch module information from. This resource will contain the address of the container registry to pull modules from, authentication parameters, and other settings.

An example of a ModuleSource resource:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: example
spec:
  registry:
    repo: registry.example.com/deckhouse/modules
    dockerCfg: <base64 encoded credentials>
```

After the ModuleSource resource is created, DKP will start to perform periodic (every three minutes) data synchronization with the module source (fetching information about the modules available in the source).

Use the following command to check the synchronization status:

```
d8 k get ms
```

If the synchronization is successful, you will see output similar to the one below:

```
$ d8 k get ms
NAME        COUNT   SYNC   MSG
example     2       16s    Ready
```

If there are synchronization errors, the `MSG` column will contain a general description of the error, e.g.:

```
$ d8 k get ms
NAME         COUNT   SYNC   MSG
example      2       16s    Some errors occurred. Inspect status for details
```

Detailed error information can be found in the `pullError` field in the status of the ModuleSource resource.

For example, here’s how you can get detailed error description from the `example` module source:

```
$ d8 k get ms example -o jsonpath='{range .status.modules[*]}{.name}{" module error:\n\t"}
{.pullError}{"\n"}{end}'
module-1 module error:
  fetch image error: GET https://registry.example.com/v2/deckhouse/modules/module-
1/release/manifests/stable: MANIFEST_UNKNOWN: manifest unknown; map[Tag:stable]
module-2 module error:
  fetch image error: GET https://registry.example.com/v2/deckhouse/modules/module-
2/release/manifests/stable: MANIFEST_UNKNOWN: manifest unknown; map[Tag:stable]
```

If synchronization is successful, the `.status.modules` field of the ModuleSource resource will contain a list of modules ready to be enabled in the cluster.

Here is an example of how you can get a list of modules available from the `example` module source:

```
$ d8 k get ms example -o jsonpath='{.status.modules[*].name}'
module-1 module-2
```

The complete list of modules available from all module sources created in the cluster can be retrieved using the following command:

```
d8 k get ms  -o jsonpath='{.items[*].status.modules[*].name}'
```

After creating the ModuleSource resource and successful synchronization, *modules* — Module resources should start appearing in the cluster (DKP creates them automatically, you do not need to create them). You can view the list of modules using the following command:

```
d8 k get module
```

Example of getting a list of modules:

```
$ d8 k get module
NAME       STAGE    SOURCE   PHASE       ENABLED   READY
module-one                   Available   False     False
module-two                   Available   False     False
```

To get additional information about the module, use the following command:

```
d8 k get module module-one -oyaml
```

Example of output:

```
apiVersion: deckhouse.io/v1alpha1
kind: Module
metadata:
  creationTimestamp: "2024-12-12T10:49:40Z"
  generation: 1
  name: module-one
  resourceVersion: "241504954"
  uid: 3ae75474-8e96-4105-a939-6df71cba82d8
properties:
  availableSources:
  - example
status:
  conditions:
  - lastProbeTime: "2024-12-12T10:49:41Z"
    lastTransitionTime: "2024-12-12T10:49:41Z"
    message: disabled
    reason: Disabled
    status: "False"
    type: EnabledByModuleConfig
  - lastProbeTime: "2024-12-12T10:49:41Z"
    lastTransitionTime: "2024-12-12T10:49:41Z"
    status: "False"
    type: EnabledByModuleManager
  - lastProbeTime: "2024-12-16T15:46:26Z"
    lastTransitionTime: "2024-12-12T10:49:41Z"
    message: not installed
    reason: NotInstalled
    status: "False"
    type: IsReady
  phase: Available
```

You can find available sources from which the module can be downloaded in the Module (there is only one in the example).

Next, you need to enable the module. To do this, you need to create a ModuleConfig with the name of the module.

The parameter `enabled` in ModuleConfig is responsible for enabling the module. If the module is available from multiple sources (resource ModuleSource), the required source can be specified in the `source` parameter.

The update policy (the name of the ModuleUpdatePolicy) can be specified in the `updatePolicy` parameter. It is not necessary to specify the update policy; in this case, it will be inherited from the Deckhouse Kubernetes Platform update parameters (releaseChannel and update of the ModuleConfig `deckhouse`).

Example of ModuleConfig for enabling the module `module-one` from the source `example`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: module-one
spec:
  enabled: true
  source: example
```

⚠ If there are mandatory parameters in the module configuration and the module is enabled without specifying them, a configuration validation error will occur. In this case, the `D8DeckhouseModuleValidationError` alert will appear, and the module will not be successfully activated.

To get more details, use the following command:

```
d8 k get mr -l module=<MODULE_NAME>
```

Make sure to specify the required configuration parameters in `ModuleConfig` according to the module's documentation.

After turning on the module, it should enter the download phase:

```
$ d8 k get module module-one
NAME        STAGE    SOURCE   PHASE         ENABLED   READY
module-one           example  Downloading   False     False
```

⚠ If the module has not entered the download phase, check the module source (ModuleSource), as the module may not be able to download.

After a successful download, the module will enter the installation phase (`Installing`):

```
$ d8 k get module module-one
NAME        STAGE    SOURCE   PHASE         ENABLED   READY
module-one           example  Installing    False     False
```

If the module was successfully installed, it will enter the `Ready` phase:

```
$ d8 k get module module-one
NAME        STAGE    SOURCE   PHASE  ENABLED  READY
module-one           example  Ready  True     True
```

Example of a Module object in the cluster when the module has been successfully installed:

```
apiVersion: deckhouse.io/v1alpha1
kind: Module
metadata:
  creationTimestamp: "2024-11-18T15:34:15Z"
  generation: 1
  name: module-one
  resourceVersion: "242153004"
  uid: 7111cee7-50cd-4ecf-ba20-d691b13b0f59
properties:
  availableSources:
  - example
  releaseChannel: Stable
  requirements:
    deckhouse: '> v1.63.0'
    kubernetes: '> v1.25.0'
  source: example
  version: v0.7.24
  weight: 910
status:
  conditions:
  - lastProbeTime: "2024-12-12T15:49:35Z"
    lastTransitionTime: "2024-12-12T15:49:35Z"
    status: "True"
    type: EnabledByModuleConfig
  - lastProbeTime: "2024-12-17T09:35:27Z"
    lastTransitionTime: "2024-12-12T15:49:39Z"
    status: "True"
    type: EnabledByModuleManager
  - lastProbeTime: "2024-12-17T09:35:27Z"
    lastTransitionTime: "2024-12-17T09:35:25Z"
    status: "True"
    type: IsReady
  - lastProbeTime: "2024-12-17T09:32:50Z"
    lastTransitionTime: "2024-12-17T09:32:50Z"
    status: "False"
    type: IsOverridden
  hooksState: 'v0.7.24/hooks/moduleVersion.py: ok'
  phase: Ready
```

In the Module, you can see the current installed version of the module, its size, the source from which it was downloaded, its dependencies, and the release channel.

In case of any errors, the module will enter the `Error` phase:

```
$ d8 k get module module-one
NAME         STAGE    SOURCE   PHASE  ENABLED  READY
module-one           example  Error  True     Error
```

If the enabled module has several available sources, and a source for the module is not explicitly selected in its ModuleConfig, the module will enter the `Conflict` phase:

```
$ d8 k get module module-one
NAME         STAGE    SOURCE   PHASE     ENABLED  READY
module-one                     Conflict  False    False
```

To resolve the conflict, specify the source of the module (ModuleSource name) explicitly in ModuleConfig.

After downloading the module, the module releases will appear in the cluster — ModuleRelease objects.

You can view the list of releases using the following command:

```
d8 k get mr
```

An example of retrieving the list of module releases:

```
$ d8 k get mr
NAME                       PHASE        UPDATE POLICY   TRANSITIONTIME   MESSAGE
module-one-v0.7.23         Superseded   deckhouse       33h
module-one-v0.7.24         Deployed     deckhouse       33h
module-two-v1.2.0          Superseded   deckhouse       48d
module-two-v1.2.1          Superseded   deckhouse       48d
module-two-v1.2.3          Deployed     deckhouse       48d
module-two-v1.2.4          Superseded   deckhouse       44d
module-two-v1.2.5          Pending      deckhouse       44d              Waiting for the
'release.deckhouse.io/approved: \"true\"' annotation
```

If the module release is in the `Superseded` status, it means that the module release is outdated, and there is a newer release that has replaced it.

⚠ If a module release is `Pending`, it means that manual confirmation is required to install it (see module update policy below). You can confirm the module release using the following command (specify the moduleRelease name):

```
d8 k annotate mr <module_release_name> modules.deckhouse.io/approved="true"
```

This can also be done using the `d8` CLI for convenience (module names and versions are autocompleted):

```
d8 system module approve <module-name> <version>
```

Switching the module to a different module source

Follow these steps to deploy a module from a different module source:

1. Create a new ModuleSource resource.
2. Specify it in the `source` field in ModuleConfig.
3. Ensure that new module releases (ModuleRelease objects) are created from the new module source in accordance with the update policy:

   ```
   d8 k get mr
   ```

Module update policy

The module update policy refers to the rules that DKP uses to update modules in the cluster. It is set by the ModuleUpdatePolicy resource with the following settings:

- module update mode (automatic, manual, updates are disabled);
- the release channel to use for updates;
- time windows for automatic updates during which the module update is permitted.

You do not have to create the ModuleUpdatePolicy resource. If the update policy for a module is not defined (there is no corresponding ModuleUpdatePolicy resource), the update settings match the update settings of DKP (the update parameter of the `deckhouse` module).

Example of the ModuleUpdatePolicy resource, whose update policy allows automatic module updates on Mondays and Wednesdays from 13:30 to 14:00 UTC:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModuleUpdatePolicy
metadata:
  name: example-update-policy
spec:
  releaseChannel: Alpha
  update:
    mode: Auto
    windows:
    - days:
      - "Mon"
      - "Wed"
      from: "13:30"
      to: "14:00"
```

The update policy is specified in the `updatePolicy` field in ModuleConfig.

Enabling the module

Before enabling the module, make sure that it can be enabled. Run the following command to list all the available DKP modules:

```
d8 k get modules
```

The module must be in the list.

Below is an example of the output:

```
$ d8 k get module
NAME       STAGE    SOURCE   PHASE       ENABLED   READY
...
module-one                   Available   False     False
module-two                   Available   False     False
...
```

It shows that the `module-one` module can be enabled.

If the module is not in the list, check that module source is defined and the module is listed in the module source. Also check the update policy of the module (if defined). If the module update policy is not defined, it matches the DKP update policy (the releaseChannel parameter and the update section of the `deckhouse` module parameters).

You can enable the module similarly to built-in DKP modules using any of the following methods:

- Run the command below (specify the name of the module):

  ```
  d8 system module enable <MODULE_NAME>
  ```

- Create a `ModuleConfig` resource containing the `enabled: true` parameter and module settings..

  Below is an example of a ModuleConfig that enables and configures the `module-one` module in the cluster:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: module-one
  spec:
    enabled: true
    settings:
      parameter: value
    version: 1
  ```

Troubleshooting

If there were errors while enabling a module in the cluster, you can learn about them as follows:

- View Deckhouse log:

  ```
  d8 k -n d8-system logs -l app=deckhouse
  ```

- View the Module object in more detail:

  ```
  d8 k get module module-one -oyaml
  ```

- View the ModuleConfig object of the module.

  Here is an example of the error message for `module-one`:

  ```
  $ d8 k get moduleconfig module-1
  NAME         ENABLED   VERSION   AGE   MESSAGE
  module-one   true                7s    Ignored: unknown module name
  ```

- View the ModuleSource object.

  Example output if the module source has problems with downloading the module:

  ```
  $ d8 k get ms
  NAME         COUNT   SYNC   MSG
  example      2       16s    Some errors occurred. Inspect status for details
  ```

Similar to DeckhouseRelease (a DKP release resource), modules have a ModuleRelease resource. DKP creates ModuleRelease resources based on what is stored in the container registry. When troubleshooting module issues, check the ModuleRelease available in the cluster as well:

```
d8 k get mr
```

Output example:

```
$ d8 k get mr
NAME                  PHASE        UPDATE POLICY          TRANSITIONTIME   MESSAGE
module-1-v1.23.2      Pending      example-update-policy  3m               Waiting for the
'release.deckhouse.io/approved: "true"' annotation
```

The example output above illustrates ModuleRelease message when the update mode (update.mode of the ModuleUpdatePolicy is set to `Manual`. In this case, you must manually confirm the installation of the new module version by adding the `modules.deckhouse.io/approved="true"` annotation to the ModuleRelease:

```
d8 k annotate mr module-1-v1.23.2 modules.deckhouse.io/approved="true"
```

This can also be done using the `d8` CLI for convenience (module names and versions are autocompleted):

```
d8 system module approve module-1 v1.23.2
```

Integrating Deckhouse Module Tools for Module Validation

To enable automatic validation of the module structure and, if needed, metrics reporting, you can integrate Deckhouse Module Tools (DMT) into your build process.

For GitHub Projects

A dedicated GitHub Action is available for integrating the DMT into your module.

To connect the DMT, add the following step to your build workflow configuration in `[project].github/workflows/build.yml`:

```
jobs:
  lint:
    runs-on: ubuntu-latest
    continue-on-error: true
    name: Linting
    steps:
      - uses: actions/checkout@v4
      - uses: deckhouse/modules-actions/lint@main
      env:
         DMT_METRICS_URL: ${{ secrets.DMT_METRICS_URL }}
         DMT_METRICS_TOKEN: ${{ secrets.DMT_METRICS_TOKEN }}
```

The `DMT_METRICS_URL` and `DMT_METRICS_TOKEN` variables are optional. If set, the DMT will send telemetry to the specified endpoint.

> **i** If the module resides in the `deckhouse` GitHub organization, these variables will be automatically populated from the configured secrets.

A complete example of the configuration can be found in the build_dev.yml file.

To simplify your setup, you can also use the provided configuration templates.

For GitLab Projects

For GitLab projects, ready-to-use templates are available and can be included in your `.gitlab-ci.yml` file to automatically configure the build and validation processes:

- **Setup**: Setup configuration template
- **Build**: Build process configuration template

Steps to connect

1. In your project's `.gitlab-ci.yml` file, add references to the templates:

   ```
   include:
     - remote: https://raw.githubusercontent.com/deckhouse/modules-gitlab-
   ci/refs/heads/main/templates/Setup.gitlab-ci.yml
     - remote: https://raw.githubusercontent.com/deckhouse/modules-gitlab-
   ci/refs/heads/main/templates/Build.gitlab-ci.yml
   ```

2. After adding the templates, in the same `.gitlab-ci.yml` configuration, add a step to perform the check:

   ```
   Lint:
     extends: .lint
   ```

Additional resources

- Global custom resources
- Module `deckhouse` parameters

Dependencies of the Deckhouse Kubernetes Platform modules

This section covers the dependencies you can set for the module.

Dependencies are a set of conditions (requirements) that must be met in order for Deckhouse Kubernetes Platform to be able to run a module.

Deckhouse Kubernetes Platform (DKP) supports the following module dependencies:

- Deckhouse Kubernetes Platform version;
- Kubernetes version;
- version of other modules.

Deckhouse Kubernetes Platform version dependency

This dependency defines the minimum or maximum DKP version with which the module is compatible.

An example of setting up a dependency for Kubernetes 1.27 and higher in the `module.yaml` file:

```
name: test
weight: 901
requirements:
   deckhouse: ">= 1.61"
```

> i For testing, you can set the `TEST_EXTENDER_DECKHOUSE_VERSION` environment variable to imitate the desired version of Deckhouse Kubernetes Platform.

Deckhouse checks whether the dependency is met in the following cases:

1. **When installing or upgrading a module** If the DKP version does not meet the requirements specified in the release module dependencies, the latter will not be installed or upgraded.

   Below is an example of the ModuleRelease resource for which the DKP version does not meet the module requirements:

   ```
   root@dev-master-0:~# d8 k get mr
   ```

   Output information:

   ```
      NAME                    PHASE        UPDATE POLICY   TRANSITIONTIME   MESSAGE
      test-v0.8.3             Pending      test-alpha      2m30s            requirements are not satisfied:
   current deckhouse version is not suitable: 1.0.0 is less than or equal to v1.64.0
   ```

2. **When upgrading Deckhouse Kubernetes Platform** Deckhouse checks if the new DKP version matches the dependencies of the installed and active modules. If at least one module is not compatible with the new version, the DKP upgrade will not be performed.

   Below is an example of the DeckhouseRelease resource for which the DKP version does not meet the module requirements:

   ```
   root@dev-master-0:~# d8 k get deckhousereleases.deckhouse.io
   ```

   Output information:

   ```
   NAME                    PHASE         TRANSITIONTIME   MESSAGE
   v1.73.3                 Skipped       74m
   v1.73.4                 Pending       2m13s            requirements of test are not satisfied: v1.73.4
   deckhouse version is not suitable: v1.73.4 is greater than or equal to v1.73.4
   ```

3. **When conducting initial module analyses** Deckhouse checks the current version of DKP and the dependencies of the installed modules. If a mismatch is discovered, the module will be disabled.

Kubernetes version dependency

This dependency defines the minimum or maximum Kubernetes version with which the module is compatible.

Here is how you can enable the Kubernetes version dependency for Kubernetes 1.28 and higher in the `module.yaml` file:

```
name: test
weight: 901
requirements:
   kubernetes: ">= 1.28"
```

> i For testing, you can set the `TEST_EXTENDER_KUBERNETES_VERSION` environment variable to imitate the desired version of Deckhouse Kubernetes Platform.

Deckhouse checks whether the dependency is met in the following cases:

1. **When installing or upgrading a module** If the Kubernetes version does not meet the requirements specified in the release module dependencies, the latter will not be installed or upgraded.

   Below is an example of the ModuleRelease resource for which the Kubernetes version does not meet the module requirements:

   ```
   root@dev-master-0:~# d8 k get modulereleases.deckhouse.io
   ```

Output information:

```
NAME                          PHASE        UPDATE POLICY   TRANSITIONTIME   MESSAGE
test-v0.8.2                   Pending      test-alpha      24m              requirements are not satisfied:
current kubernetes version is not suitable: 1.29.6 is less than or equal to 1.29
virtualization-v.0.0.0-dev4   Deployed      deckhouse      142d
```

2. **When upgrading Kubernetes** Deckhouse examines the dependencies of active modules, and if at least one module is incompatible with the new Kubernetes version, the version upgrade will not proceed.

   Below is an example of the output you may encounter when a module is incompatible with a newer version of Kubernetes:

   ```
   root@dev-master-0:~# d8 system edit cluster-configuration
   ```

   Output information:

   ```
   Save cluster-configuration back to the Kubernetes cluster
   Update cluster-configuration secret
   Attempt 1 of 5 |
           Update cluster-configuration secret failed, next attempt will be in 5s"
           Error: admission webhook "kubernetes-version.deckhouse-webhook.deckhouse.io" denied the request:
   requirements of test are not satisfied: 1.27 kubernetes version is not suitable: 1.27.0 is less than or
   equal to 1.28
   ```

3. **When conducting initial module analyses** If the Kubernetes version does not conform to the dependencies of the modules that are already installed, DKP will disable those modules.

4. **When upgrading Deckhouse Kubernetes Platform** Deckhouse checks the default Kubernetes version value for DKP and if it is not compatible with the active modules, the DKP update will not be carried out.

   Below is an example of the DeckhouseRelease resource for which the Kubernetes version does not meet the module requirements:

   ```
   root@dev-master-0:~# d8 k get deckhousereleases.deckhouse.io
   ```

   Output information:

   ```
   NAME                    PHASE         TRANSITIONTIME   MESSAGE
   v1.73.3                 Pending       7s              requirements of test are not satisfied: 1.27
   kubernetes version is not suitable: 1.27.0 is less than or equal to 1.28
   ```

Dependency on the version of other modules

Dependencies on other modules describe the conditions for enabling, updating, and disabling a module. A module in the Deckhouse Kubernetes Platform may have required and conditional dependencies on versions of other modules.

Required dependencies

This dependency defines the list of **enabled** modules and their versions that are required for the module to work.

> **i** The built-in DKP module version is considered equal to the DKP version.

If you need to specify that some module is simply enabled, no matter what version, then you can use the following syntax (using the `user-authn` module as an example):

```
requirements:
  modules:
    user-authn: ">= 0.0.0"
```

Example of setting up a dependency on three modules:

```
name: hello-world
requirements:
  modules:
    ingress-nginx: '> 1.67.0'
    node-local-dns: '>= 0.0.0'
    operator-trivy: '> v1.64.0'
```

Conditional module requirements

✖ Conditional dependencies for modules are available for Deckhouse Kubernetes Platform starting with version 1.73. If you need to use them for a module, set a Deckhouse Kubernetes Platform version dependency 1.73 or higher for that module.

Use this when your module works alone, but integrates with another module **if it is present**.

i Conditional dependencies may affect the ability to enable, disable, and update both modules: the dependent module and the module on which it depends.

To mark a requirement as conditional, add `!optional` to the version constraint string:

```
name: prometheus
requirements:
  modules:
    test: ">v0.22.1 !optional"
```

i The following sections describes restrictions on the use of conditional module dependencies and provides examples of settings where:

- `prometheus` — the target module for which an conditional dependency is specified;
- `test` — a module that can be used in conjunction with the target module.

Restrictions on enabling and disabling prometheus when there is an conditional dependency on test

When there is an conditional dependency on the version of another module, the target module has the following restrictions on enabling and disabling:

1. If `test` is not enabled in the cluster, `prometheus` can be enabled.

   Example: `test` is disabled + `prometheus` has an conditional requirement `test: ">v0.22.1 !optional"` → `prometheus` will be enabled, the requirement is skipped.

2. If `test` is already enabled in the cluster, enabling `prometheus` with `requirements` is only possible if `requirements` specifies requirements that the current version of `test` meets.

   Example: `test` version `v0.21.1` is enabled in the cluster + an conditional requirement `test: ">v0.22.1 !optional"` is set for `prometheus` → installing/enabling `prometheus` will result in a dependency mismatch error (the current version of `test` does not match `requirements`).

3. If `test` is disabled in the cluster, `prometheus` will remain enabled.

   Example: `test` is enabled in the cluster + an conditional requirement `test: ">v0.22.1 !optional"` is set for `prometheus` → disabling `test` will be successful, `prometheus` will not be disabled.

Restrictions on updating prometheus when there is an conditional dependency on test

When there is an conditional dependency on the version of another module, the target module has the following update restrictions:

1. `prometheus` can be updated even if the cluster does not have the `test` module.

   Example: `test` is disabled + `prometheus` has an conditional requirement `test: ">v0.22.1 !optional"` → `prometheus` will be updated.

2. The update for `prometheus` will be blocked until `test` specified in `requirements` is updated in the cluster.

   Example: `test` is enabled + an conditional requirement `test: ">v0.22.1 !optional"` is set for `prometheus` → `prometheus` will not be updated until `test` is updated.

Restrictions on enabling test, whose version is specified in prometheus dependencies

If the `prometheus` module is enabled, it is not possible to enable `test`, whose version does not match the expression specified in `requirements` for `prometheus`.

Example: `prometheus` is included + an conditional requirement `test: ">v0.22.1 !optional"` is specified for `prometheus` → an attempt to include `test v0.21.1` will result in a dependency mismatch error (the version `test v0.21.1` does not match the condition in `requirements` for `prometheus`).

Restrictions on updating test, whose version is specified in prometheus dependencies

If prometheus and test are included in the cluster, test can only be updated to a version that meets the requirements specified in requirements for prometheus.

Example: The `prometheus` and `test` modules are included in the cluster + an conditional requirement `test: '=v0.22.1 !optional'` is set for `prometheus` + an attempt to update `test` to version `0.23.1` → `test` will not be updated because the required version does not meet the `requirements` for `prometheus`.

⚠
- Enabling or disabling modules may take longer because of extra extender checks.
- Known limitation: during reconciliation the list of enabled modules may briefly be empty, which in rare cases lets an conditional dependency check pass incorrectly. If this happens, retry the operation.

Versioning modules

Deckhouse Kubernetes Platform (DKP) uses semantic versioning for modules.

When choosing a version, stick to the following guidelines:

- updating the **patch version** (e.g., from `0.0.1` to `0.0.2`) means that some issue has been fixed;
- updating the **minor version** (e.g., from `0.0.1` to `0.1.0`) means that some new feature has been added;
- updating the **major version** (e.g., from `0.0.1` to `1.0.0`) means that a feature has been added that radically changes the module's capabilities; the interface has undergone significant changes or a major phase of development has been completed.

The git tag and registry container **always** have a "v" before the version number, e.g., `v0.0.73` or `v1.0.0`.

Release channels

Release channels allow you to publish a module version not for all users at once, but for a limited group of users. You define the stability level of the module version and the maximum release channel to which it can be published.

Note that the choice of a specific release channel does not determine how stable the module is. The channels are an instrument for delivery of updates and determine the degree of stability of a particular release.

When publishing a new version of the module to the release channel, first use the `Alpha` release channel. Then, if module version has no issues, publish the module version sequentially to other release channels, taking their stability into account: `Alpha` → `Beta` → `Early Access` → `Stable` → `Rock Solid`. If the module version requires fixing, the publication of that version should be halted. After releasing a version with fixes, the publication stage must be repeated starting from the `Alpha` release channel.

Procedure for publishing a new version

The recommended sequence of publication of the module version in release channels:

1. Publish a new version of the module in the `Alpha` release channel.
2. If the version works stably, publish it in the following release channels sequentially: `Beta` → `Earlyaccess` → `Stable` → `Rocksolid`.
3. If errors arise, stop the publication and correct them.
4. Repeat the publication of the version, starting from the release channel `alpha`.

Module lifecycle

During its life cycle, the module goes through the following stages:

- **Experimental** refers t to an experimental version. The module functionality may undergo significant changes. Compatibility with future versions is not guaranteed.

  ℹ Modules in the `Experimental` stage cannot be enabled by default. To allow the use of such modules, set the allowExperimentalModules parameter to true.

- **Preview** refers to a preliminary version. The module functionality may change, but the basic features will be preserved. Compatibility with future versions is ensured, but may require additional migration actions.

- **General Availability (GA)** refers to a generally available version. The module is ready to be used in production environments.
- **Deprecated** refers to a module version that has been deprecated. Further development and support of the module is discontinued.

Definition of module stability

Depending on the stage of the module lifecycle and the release channel from which the specific module version was installed, the overall stability can be determined by the following table:

Lifecycle Stage	Release Channels
	Alpha
Experimental	Experiments
Preview	Experiments
General Availability	Experiments

Lifecycle Stage	Release Channels
	Alpha
Deprecated	Deprecated

- **Experiments** — Functionality checks, experiments, and testing;
- **Beta testing** — Functionality checks, experiments, and testing. Limited use by experienced users in environments equivalent to production;
- **Limited use** — Development environments, pilot projects, low-impact production environments;
- **Production use** — Production environments and those equivalent to them;
- **Production use in critical systems** — Mission-critical production environments and those equivalent to them;
- **Deprecated** — Should be phased out of use.

Highlights:

- A module in `Experimental` status on the `Stable` channel is recommended to be used in production environments only with limitations.
- A module in `General Availability` status on the `Alpha` channel is also not recommended for use in production environments.
- Only `GA` modules installed from `Early Access`, `Stable`, or `Rock Solid` channels are suitable for production environments.
- `Deprecated` modules are recommended to be replaced.

API versioning

Modules in DKP use custom resources to interact with users. The `apiVersion` parameter with the API version of these resources is set according to the following rules:

- `v1alphaX` refers to an API that has just been published. This API needs to be tested to see how user-friendly it is, as well as how valid and consistent its settings are.
- `v1betaX` refers to the API that has passed initial testing. Its logical development and refinement is in progress.
- `v1stableX` refers to a stable API. At this point, its fields are kept in the specification while its validation rules do not change to be more stringent.

If necessary, a new version of API v2 can be released, which goes through the same stages but with the prefix `v2`. It's important to remember that after the release of the `v1stableX` version, Kubernetes will consider it more prioritized than the `alpha` or `beta` versions until a new stable version `v2stableX` is released. When executing `kubectl apply` and `kubectl edit` commands, the `v1stableX` version will be used by default.

Release of the new API version

The reasons for releasing a new version may be as follows:

- structure changes;
- updating obsolete parameters.

You can add new parameters without changing the version.

Automatic API Conversion

To enable automatic conversion of module parameters from one version to another, you can include the appropriate module conversions. Conversions may be necessary when a parameter is renamed or moved to a different location in a new version of the OpenAPI specification.

Recommendations for Releasing New Versions of CRD

Please follow these recommendations when releasing a new version of the CustomResourceDefinition (CRD):

- Set the parameter `deprecated: true` for previous versions of the CRD. For more details on working with deprecated versions of CRD, refer to the Kubernetes documentation.
- Do not change the storage version, in which data is stored inside etcd, until at least two months have passed since the release of the new version.

Additional resources

- Module `deckhouse` parameters

An example of creating a module with adaptation of an existing chart

This section provides an example of creating a `helloworld` module based on module template and an adaptation of the hello-world Helm-chart.

Preparing module source code and building

1. Install the tools you will need:
   - git
   - sed
   - yq
2. Fork the module template repository or copy it.

   ```
   git clone git@github.com:deckhouse/modules-template.git helloworld \
     && cd helloworld
   ```

3. Enter the name of the module in the `module.yaml` file.

   We will use the `helloworld` module name in this example, but you can use any name you like. For this, replace ‘helloworld’ in the commands and in your repo’s name with the one you prefer.

   > **i** Note that the module name may differ depending on the command. In may be written in *kebab-case* as well as *camelCase*. If you use a custom module name, make sure to modify it accordingly.

   Run the following command to add the module name to the `module.yaml` file or edit it manually:

   ```
   sed -i -e 's/^name:.*$/name: helloworld/' module.yaml
   ```

4. Clone the hello-world chart source code into a temporary directory.

   ```
   git clone https://github.com/giantswarm/hello-world-app .tmp-chart
   ```

5. Copy the chart templates to the `templates` directory of the module (make sure to empty it beforehand).

   ```
   rm -rf templates/*
   cp -fR .tmp-chart/helm/hello-world/templates/* templates/
   cp .tmp-chart/helm/hello-world/values.yaml values.yaml
   ```

6. Replace the `.Values` path in the chart templates with `.Values.helloworld`.

   > **i** This is due to the architectural feature of addon-operator. You have to stick to it to be able to access module values.

   ```
   sed -i -e 's/.Values/.Values.helloworld/g' $(find templates/ -type f)
   ```

7. Add the OpenAPI schema of the module settings.

   The module parameters are specified in the OpenAPI schema in the openapi directory. Execute the following command to convert the JSON schema of the chart parameters to the OpenAPI schema of the module:

   ```
   jq 'walk(
      if type == "object" and .type == "object" and (keys | length) == 1
      then . + {additionalProperties: true}
      else .
      end
   )' .tmp-chart/helm/hello-world/values.schema.json > openapi/config-values.yaml
   ```

8. Define a rule for building an application container image.

   The rules for building application container images must reside in a subdirectory of the images directory of the module. Run the following commands to create an application image directory and a Dockerfile with

the image build rules:

```
rm -rf images/*
mkdir images/helloworld
echo "FROM quay.io/giantswarm/helloworld:0.2.0" > images/helloworld/Dockerfile
```

9. Replace the image in the Deployment manifest with the Deckhouse Kubernetes Platform library helper. This will allow you to use the current content-based image tag.

```
sed -Ei 's/image\:(.*)/image: {{ include "helm_lib_module_image" (list . "helloworld")
}}/g' templates/deployment.yaml
```

10. Delete module hooks, CRDs, and temporary files.

    In our example, hooks and CustomResourceDefinitions are not used. Run the following commands to clear the `hooks` and `crds` directories:

```
rm -rf hooks/
rm -rf crds/
rm -rf .tmp-chart
```

11. Set up CI/CD.

    The project template in the `.github` directory contains pre-built GitHub Actions workflow files that implement a simple scheme for building and publishing a module using the GitHub Packages registry (ghcr.io). Module artifacts will be pushed to `ghcr.io/<OWNER>/modules/`, which will be the module source. Make changes to the workflow files if the suggested option does not work for you.

    Do the following in the properties of your project on GitHub to make the module workflow run smoothly:

    - Open the *Settings -> Actions -> General* page.
    - Enable *Read and write permissions* in the *Workflow permissions* section.

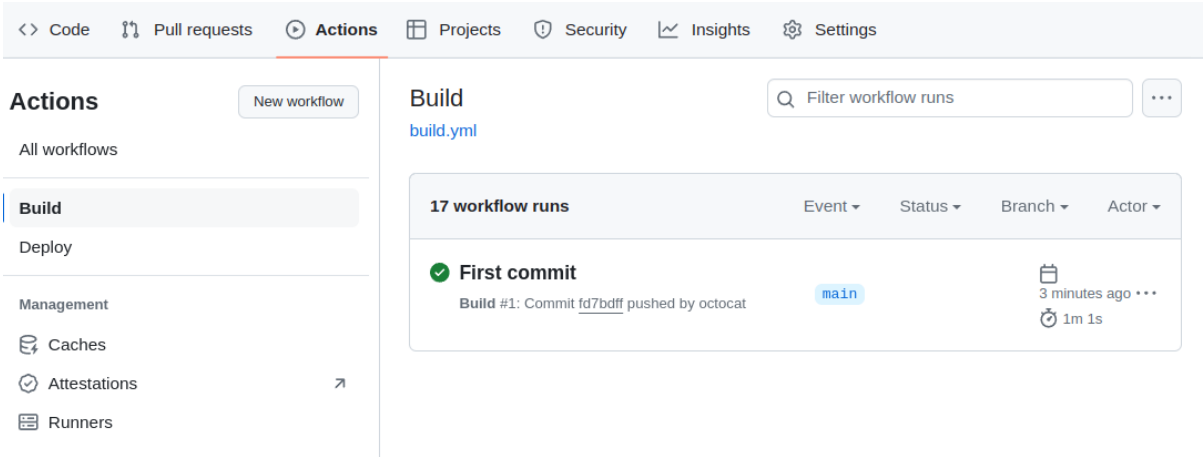
12. Commit the changes to the repository (specify the address of the module's Git repository).

```
git add .
git commit -m "Initial Commit"
git push --set-upstream origin <GIT_REPO_URL>
```

13. Confirm that the module was built successfully.

    Go to the *Actions* section of the module repository and select *Build* from the workflow list on the left. The workflow that starts after you have executed the `git push` command at the previous step is expected to run successfully.

    An example:

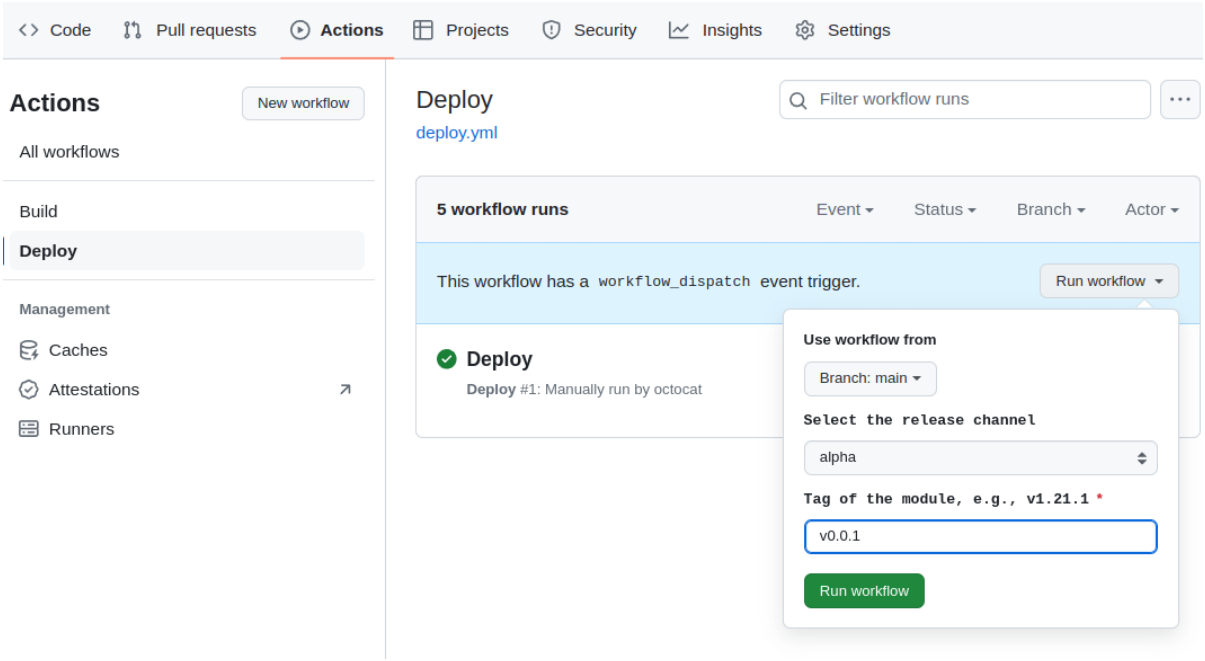


Publishing the module to the release channel

Here's how you can publish version `v0.0.1` of the module to the *Alpha* release channel:

1. Create a new release `v0.0.1` of the module in the GitHub repository or add the `v0.0.1` tag.
2. Go to the *Actions* section of the module repository and select *Deploy* in the workflow list on the left.
3. Click on the *Run workflow* drop-down list on the right of the page and select `alpha`. Enter the `v0.0.1` tag

in the tag input field. Click the *Run workflow* button.



4. Confirm that the module publishing workflow has completed successfully.

You can now access your module in a cluster managed by Deckhouse Kubernetes Platform.

Enabling a module in a cluster

The following is the sequence of steps to enable the `helloworld` in a cluster managed by Deckhouse Kubernetes Platform.

1. Create an access token in the GitHub repository with permissions to handle GitHub Packages.
2. Generate an authentication string to access the GitHub Packages container registry in dockerconfigjson format; insert your GitHub username (or organization) and access token in the angle brackets below:

```
base64 -w0 <<EOF
{
  "auths": {
    "ghcr.io": {
      "auth": "$(echo -n '<OWNER>:<TOKEN>' | base64 -w0)"
    }
  }
}
EOF
```

3. Create a ModuleSource resource in the cluster (specify the container registry address and authentication string).

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: ghcr
spec:
  registry:
    # Insert your GitHub username (or organization), e.g., ghcr.io/octocat/modules.
    repo: ghcr.io/<!OWNER>/modules
    # Paste the authentication string for accessing GitHub Packages that you got in
the previous step
    dockerCfg: <!REGISTRY_CREDENTIALS>
EOF
```

Note that it may take a few seconds to synchronize data after a resource has been created.

4. Check the list of available modules

```
$ d8 k get module
NAME       STAGE    SOURCE   PHASE       ENABLED   READY
...
helloworld                   Available   False     False
...
```

5. Create a ModuleUpdatePolicy resource that defines the module update policy.

   Run the following command to create an update policy with the *Alpha* release channel and *Auto* update mode:

   ```
   d8 k apply -f - <<EOF
   apiVersion: deckhouse.io/v1alpha2
   kind: ModuleUpdatePolicy
   metadata:
     name: helloworld-policy
   spec:
     releaseChannel: Alpha
     update:
       mode: Auto
   EOF
   ```

6. Create a ModuleConfig where you specify the module source (the `source` parameter), the update policy (the `updatePolicy` parameter), and set the `enabled` parameter to `true`:

   ```
   d8 k apply -f - <<EOF
      apiVersion: deckhouse.io/v1alpha1
      kind: ModuleConfig
      metadata:
        name: helloworld
      spec:
         enabled: true
         source: ghcr
         updatePolicy: helloworld-policy
   ```

7. Check the ModuleSource (the status should have no errors and list the available modules):

   ```
   d8 k get ms ghcr -o yaml
   ```

8. Confirm that the new ModuleRelease objects have been created for the module:

   ```
   d8 k get mr
   ```

   An example of output:

   ```
   $ d8 k get mr
   NAME                                PHASE        UPDATE POLICY        TRANSITIONTIME
   MESSAGE
   helloworld-v0.0.1                   Deployed     helloworld-policy    22m
   ```

9. If the release has been successfully installed, wait for the Deckhouse Kubernetes Platform pod to restart.

   ```
   d8 k -n d8-system get pod -l app=deckhouse
   ```

   After a while, the module objects will be available in the cluster.

   If you run into errors while starting the module, check the DKP log:

   ```
   d8 k -n d8-system logs deploy/deckhouse -f | jq -rc '.msg'
   ```

or check the status of the DKP queue:

```
d8 system queue list
```

Migration of ModuleUpdatePolicy to version v1alpha2

If there is a ModuleUpdatePolicy of version v1alpha1 in the cluster, the following migration steps to version v1alpha2 must be performed:

If any ModuleUpdatePolicy of version v1alpha1 in the cluster defines `moduleReleaseSelector`, alerts `ModuleHasDeprecatedUpdatePolicy` will be triggered in the monitoring system for all modules that match this selector. In this case, follow these steps to migrate to version v1alpha2 of ModuleUpdatePolicy:

- Specify the update policy for the corresponding modules in the ModuleConfig `spec.updatePolicy` parameter.
- Execute the following command, specifying the required ModuleUpdatePolicy:

```
d8 k patch moduleupdatepolicies.v1alpha1.deckhouse.io <MUP_NAME> --type='json' \
  -p='[{"op": "replace", "path":
"/spec/moduleReleaseSelector/labelSelector/matchLabels", "value": {"": ""}}]'
```

Additional resources

- Global custom resources

Module development and debugging

When developing modules, you may want to pull and deploy a module bypassing the release channels. The ModulePullOverride resource is used for this purpose.

> ⚠ The ModulePullOverride resource is intended **for development and debugging environments only**. Using it in production clusters is **not recommended*. Support for the resource might be removed in future Deckhouse Kubernetes Platform versions.

An example of ModulePullOverride:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModulePullOverride
metadata:
  name: <module-name>
spec:
  imageTag: <tag of the module image>
  scanInterval: <image digest check interval. Default: 15s>
```

The `maintenance mode` can also be used for development purposes. In this mode, Deckhouse disables resource management for the module and does not apply changes automatically. This mode is not intended for use in production clusters.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
...
spec:
  enabled: true
  maintenance: NoResourceReconciliation
  settings:
```

Requirements for the resource parameters:

- The module name (`metadata.name`) must match the module name in the ModuleSource (`.status.modules.[].name`).
- The container image tag (`spec.imageTag`) can be anything, e.g., `pr333`, `my-branch`.

The `spec.scanInterval` time interval (optional) defines the interval for scanning images in the registry. The default interval is 15 seconds. To force scan you can change the interval or set the `renew=""` annotation on ModulePullOverride.

The `spec.rollback` indicates whether the deployed module release should be rollback after deleting the ModulePullOverride.

You can get the result of applying ModulePullOverride in the message (column `MESSAGE`) when retrieving ModulePullOverride information. The value `Ready` indicates the successful application of ModulePullOverride parameters. Any other value indicates conflict.

Example of absence of conflicts when using ModulePullOverride:

```
$ d8 k get modulepulloverrides.deckhouse.io
NAME      UPDATED   MESSAGE   ROLLBACK
example1  10s       Ready     false
```

Requirements for the module:

- The module must exist; otherwise the message for ModulePullOverride will be *The module not found*.

  An example:

  ```
  $ d8 k get modulepulloverrides.deckhouse.io
  NAME      UPDATED   MESSAGE                ROLLBACK
  example1  10s       The module not found   false
  ```

- The module must not be embedded Deckhouse module; otherwise the message in ModulePullOverride will be *The module is embedded*.

  An example:

  ```
  $ d8 k get modulepulloverrides.deckhouse.io
  NAME            UPDATED  MESSAGE                  ROLLBACK
  ingress-nginx   10s      The module is embedded   false
  ```

- The module must be enabled; otherwise, the message for ModulePullOverride will be *The module disabled*.

  An example:

  ```
  $ d8 k get modulepulloverrides.deckhouse.io
  NAME     UPDATED   MESSAGE               ROLLBACK
  example  7s        The module disabled   false
  ```

- The module must have a source; otherwise the message at ModulePullOverride will be *The module does not have an active source*.

  An example:

  ```
  $ d8 k get modulepulloverrides.deckhouse.io
  NAME       UPDATED   MESSAGE                                    ROLLBACK
  example    12s       The module does not have an active source   false
  ```

- The source for the module must exist; otherwise the message for ModulePullOverride will be *The source not found*.

  An example:

  ```
  $ d8 k get modulepulloverrides.deckhouse.io
  NAME       UPDATED   MESSAGE                 ROLLBACK
  example    12s       The source not found    false
  ```

To update the module without waiting for the next update cycle to begin, you can execute the following command:

```
d8 k annotate mpo <name> renew=""
```

Module availability and enabling by default

To assign Deckhouse editions the module should be available in, as well as module bundles where it should be enabled by default, use the `accessibility` field in `module.yaml`:

```
name: test
accessibility:
  editions:
    ee:
      available: true
      enabledInBundles:
        - Default
```

In this configuration, the module will be available in `ee` (DKP Enterprise Edition)
and can be enabled using the ModuleConfig object, and will be enabled by default in the `Default` bundle.

How ModulePullOverride works

After creating ModulePullOverride, the corresponding module will not consider ModuleUpdatePolicy, and will also not load and create ModuleRelease objects. The module will be loaded upon every change of the `imageDigest` parameter, after which it will be applied in the cluster. The ModuleSource status will have `overridden: true`, which indicates that ModulePullOverride is being used instead of ModuleUpdatePolicy. Also, the corresponding Module object will have an `IsOverridden` field in its status, and the module version from `imageTag`.

An example:

```
apiVersion: deckhouse.io/v1alpha1
kind: Module
metadata:
  creationTimestamp: "2024-11-18T15:34:15Z"
  generation: 16
  labels:
    deckhouse.io/epoch: "1326105356"
  name: example
  resourceVersion: "230347744"
  uid: 7111cee7-50cd-4ecf-ba20-d691b13b0f59
properties:
  availableSources:
  - example
  releaseChannel: Stable
  requirements:
    deckhouse: '> v1.63.0'
    kubernets: '> v1.30.0'
  source: example
  version: mpo-tag
  weight: 910
status:
  conditions:
  - lastProbeTime: "2024-12-03T15:57:20Z"
    lastTransitionTime: "2024-12-03T15:57:20Z"
    status: "True"
    type: EnabledByModuleConfig
  - lastProbeTime: "2024-12-03T15:59:58Z"
    lastTransitionTime: "2024-12-03T15:57:26Z"
    status: "True"
    type: EnabledByModuleManager
  - lastProbeTime: "2024-12-03T15:59:58Z"
    lastTransitionTime: "2024-12-03T15:56:23Z"
    status: "True"
    type: IsReady
  - lastProbeTime: "2024-12-03T15:59:48Z"
    lastTransitionTime: "2024-12-03T15:56:47Z"
    status: "True"
    type: IsOverridden
  phase: Ready
```

The module will keep running after ModulePullOverride is removed. But if there is a ModuleUpdatePolicy for the module, new releases of the module (ModuleRelease) will be pulled to replace the current “developer version”.

An example

1. Suppose there are two modules, `echo` and `hello-world`, defined in ModuleSource. The update policy is set for them, and they are pulled in and installed in DKP:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: test
spec:
  registry:
    ca: ""
    dockerCfg: someBase64String==
    repo: registry.example.com/deckhouse/modules
    scheme: HTTPS
status:
  modules:
  - name: echo
    policy: test-alpha
  - name: hello-world
    policy: test-alpha
  modulesCount: 2
```

2. Enable the module and create ModulePullOverride for the `echo` module:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModulePullOverride
metadata:
  name: echo
spec:
  imageTag: main-patch-03354
```

After creating ModulePullOverride, the image tag `registry.example.com/deckhouse/modules/echo:main-patch-03354` will be used for the module (`ms:spec.registry.repo/mpo:metadata.name:mpo:spec.imageTag`).

1. The ModulePullOverride will change with each update of the module:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModulePullOverride
metadata:
  name: echo
spec:
  imageTag: main-patch-03354
  scanInterval: 15s
status:
  imageDigest: sha256:ed958cc2156e3cc363f1932ca6ca2c7f8ae1b09ffc1ce1eb4f12478aed1befbc
  message: "Ready"
  updatedAt: "2023-12-07T08:41:21Z"
```

   where:

   - `imageDigest` is the unique identifier of the container image that was pulled.
   - `lastUpdated` is the time when the image was last pulled.

2. In this case, ModuleSource would look as follows:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: test
spec:
  registry:
    ca: ""
    dockerCfg: someBase64String==
    repo: registry.example.com/deckhouse/modules
    scheme: HTTPS
status:
  modules:
  - name: echo
    overridden: true
  - name: hello-world
    policy: test-alpha
  modulesCount: 2
```

Module auto-update logic



> **i** ModuleRelease versions v1.0.0 and v1.1.1 are provided as examples.

1. **Module installation**. When a module is enabled (`enable module <module name>`), the latest available version from the selected stability channel is automatically downloaded and deployed to the cluster. For example, this could be ModuleRelease v1.0.0. The most recent version is used; older versions are not installed.
2. **Module disabling**. When a module is disabled (`disable module <module name>`):
   - The module stops receiving new releases.
   - The currently deployed version remains in the cluster with the `Deployed` status.
3. **Behavior on re-enabling**.

   If the module is re-enabled within 72 hours:

   - The previously deployed version (ModuleRelease v1.0.0) is used.
   - New releases are checked.
   - If available, they are downloaded (e.g., v1.1.0, v1.1.1).
   - The module is then updated according to the standard update rules (Update). More information

   If the module is re-enabled after 72 hours:

   - The old version is deleted (`delete ModuleRelease v1.0.0`).
   - Upon re-enabling, the latest available version is downloaded (e.g., v1.1.1).
   - The cycle starts again as if the module was enabled for the first time (see step 1).
4. **Behavior of a disabled module**. If a module is disabled, no new releases are downloaded. The previously deployed version (the last one that was enabled) is removed after 72 hours if the module is not re-enabled within that time.

Skipping intermediate releases (from-to)

The `from-to` mechanism allows you to skip step-by-step module updates. If the current installed module version (status `Deployed`) falls within the `from-to` range, DKP skips intermediate releases and installs the latest available version within `to`.

To enable the mechanism, set transition rules in the module configuration (`module.yaml`). Example:

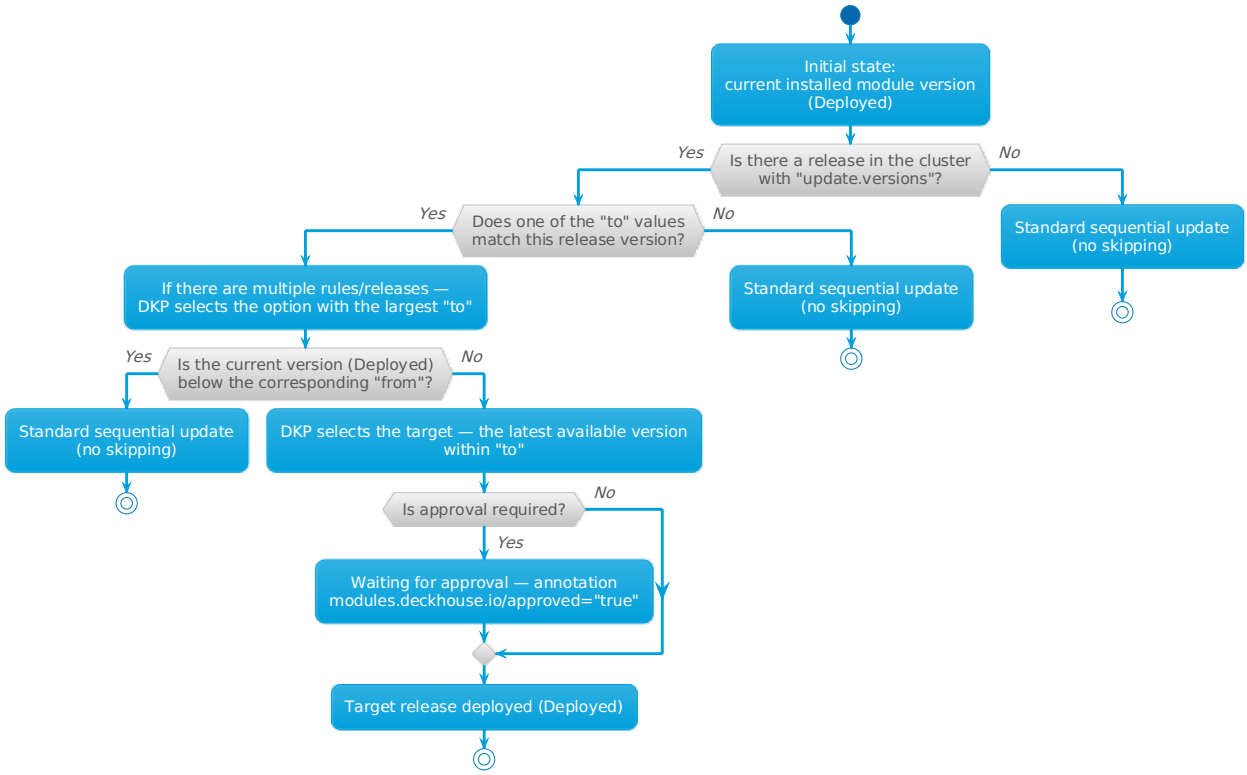
```
update:
  versions:
    - from: "1.67"
      to:   "1.75"
    - from: "1.99"
      to:   "2.0" # Transitions between major versions are specified in the X.Y format.
```

> **i** A constrained release is a module release whose `module.yaml` contains the `update.versions` section. The `from-to` mechanism works only with such releases.

Conditions for applying `from-to`:

- Target release version. The rule is read from the release you want to move to — the `to` value must match the version of that very release (constrained release).
- Current version threshold. The current installed module version (`Deployed`) is not lower than `from`. If the current version is lower than `from`, the rule does not apply — the update proceeds sequentially.
- If multiple releases match at the same time, the option with the largest `to` is chosen (the rules may reside in different ModuleRelease objects of the same module).
- If no release meets these conditions, the update proceeds as usual — without skipping intermediate versions.

If a release with update.versions appears in the cluster, DKP does not require updating in order — such a release appears “as is” in the list, DKP automatically selects a suitable option and, if necessary, waits for approval. You can immediately approve installation of the latest available version within `to`. After approval, intermediate releases between `from` and `to` get the `Skipped` status after reconciliation (not immediately); for some time between `Superseded` and `Deployed` there may be releases in the `Pending` status.



Check available releases (ModuleRelease) with:

```
d8 k get mr
```

Example output if the current module version is `0.3.33`, and `module.yaml` contains the rule `from: "0.3" → to: "0.7"`:

```
p-o-test-v0.1.0    Superseded
p-o-test-v0.2.22   Superseded
p-o-test-v0.3.33   Deployed
p-o-test-v0.4.1    Pending      Release is waiting for the 'modules.deckhouse.io/approved:
"true"' annotation
p-o-test-v0.5.27   Pending      awaiting for v0.4.1 release to be deployed
p-o-test-v0.6.11   Pending      awaiting for v0.4.1 release to be deployed
p-o-test-v0.7.25   Pending      Release is waiting for the 'modules.deckhouse.io/approved:
"true"' annotation
```

In this example output, the current installed version is `0.3.33` (status `Deployed`). According to the rule `from: 0.3 → to: 0.7`, the latest available version within `0.7` is selected for installation — `p-o-test-v0.7.25` (status `Pending`). After approval, versions `0.4.1`, `0.5.27`, and `0.6.11` change to `Skipped`, and `p-o-test-v0.7.25` becomes `Deployed`.

To approve the module release, apply the annotation:

```
d8 k annotate mr p-o-test-v0.7.25 modules.deckhouse.io/approved="true"
```

This can also be done using the `d8` CLI for convenience (module names and versions are autocompleted):

```
d8 system module approve p-o-test v0.7.25
```

Example output after approval:

```
p-o-test-v0.4.1    Skipped
p-o-test-v0.5.27   Skipped
p-o-test-v0.6.11   Skipped
p-o-test-v0.7.25   Deployed
```

Whether an annotation is required depends on the module's update policy. More details — Module update policy.

Examples

Example 1. The target release contains the rule, and its version equals `to` — the transition is performed.

In this example, the `from-to` rule is defined in the **target** release, and the `to` value equals the version of that release. The current installed version (`Deployed`) is not lower than `from`, so the transition is possible. As soon as the release appears in the cluster, DKP prepares the update. If the update policy requires approval, mark the release with the annotation. After approval, intermediate releases will be skipped and the target release becomes `Deployed`:

```
# Module version (Deployed): v1.67.23 (≥ 1.67) → the 'from' condition is satisfied.
metadata:
  name: module-test-v1.67.23
status:
  phase: Deployed

# Target release: the release version equals update.versions.to (1.75).
metadata:
  name: module-test-v1.75.25
update:
  versions:
  - from: "1.67"
    to:   "1.75"   # The update will be performed to this version after approval (if
required).
```

Likewise, if the current version is even higher than `from`, the transition is also performed:

```
# Module version (Deployed): v1.69.0  (≥ 1.67) → the 'from' condition is satisfied.
metadata:
  name: module-test-v1.69.0
status:
  phase: Deployed

# Target release: the release version equals update.versions.to (1.75).
metadata:
  name: module-test-v1.75.25
update:
  versions:
  - from: "1.67"
    to:   "1.75"  # The update will be performed to this version after approval (if
required).
```

Example 2. The current version is lower than `from` — the transition is not performed.

If the current installed version is lower than `from`, the rule does not apply. The update proceeds sequentially (without skipping):

```
# Module version (Deployed): v1.61.0  (< 1.67) → the 'from' condition is NOT satisfied.
metadata:
  name: module-test-v1.61.0
status:
  phase: Deployed

metadata:
  name: module-test-v1.75.25
update:
  versions:
  - from: "1.67"
    to:   "1.75"   # The from-to transition is not performed → sequential update.
```

Example 3. `to` does not match the version of the release that contains the rule — the transition is not performed.

A transition is possible only to a release where `to` equals the version of that release. If the rule is described in another release (for example, `to: "1.74"` is placed in release `v1.75.25`), it will not work — the update goes sequentially.

```
# Module version (Deployed): v1.67.0.
metadata:
  name: module-test-v1.67.0
status:
  phase: Deployed

metadata:
  name: module-test-v1.74.0  # This release has no update.versions.

# The rule is written in another release (v1.75.25), and 'to' does not equal its version.
metadata:
  name: module-test-v1.75.25
update:
  versions:
  - from: "1.67"
    to:   "1.74"   # 'to' does not equal this release version (1.75.25) → the rule is
ignored.
```

Module artifacts in the container registry

After a module has been built, its artifacts must be pushed to the container registry at a path that is the *source* path for pulling and running modules in DKP. The path where module artifacts are pushed to the registry is specified in the ModuleSource resource.

Below is an example of the container image hierarchy after pushing the `module-1` and `modules-2` module artifacts into the registry:

```
registry.example.io
 modules-source
├─  module-1
│  ├─  v1.23.1
│  ├─  d4bf3e71015d1e757a8481536eeabda98f51f1891d68b539cc50753a-1589714365467
│  ├─  e6073b8f03231e122fa3b7d3294ff69a5060c332c4395e7d0b3231e3-1589714362300
│  ├─  v1.23.2
│  └─  release
│     ├─  v1.23.1
│     ├─  v1.23.2
│     ├─  alpha
│     └─  beta
└─  module-2
   ├─  v0.30.147
   ├─  d4bf3e71015d1e757a8481536eeabda98f51f1891d68b539cc50753a-1589714365467
   ├─  e6073b8f03231e122fa3b7d3294ff69a5060c332c4395e7d0b3231e3-1589714362300
   ├─  v0.31.1
   └─  release
      ├─  v0.30.147
      ├─  v0.31.1
      ├─  alpha
      └─  beta
```

> ⚠ The container registry must support a nested repository structure. See the requirements section for more details.

Below is a list of commands for working with the module source. The examples use the crane tool. Follow the instructions to install it. For macOS, use `brew`.

Print the list of modules in the module source

```
crane ls <REGISTRY_URL>/<MODULE_SOURCE>
```

An example:

```
$ crane ls registry.example.io/modules-source
module-1
module-2
```

Print the list of module images

```
crane ls <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>
```

An example:

```
$ crane ls registry.example.io/modules-source/module-1
v1.23.1
d4bf3e71015d1e757a8481536eeabda98f51f1891d68b539cc50753a-1589714365467
e6073b8f03231e122fa3b7d3294ff69a5060c332c4395e7d0b3231e3-1589714362300
v1.23.2
```

In the example above, there are two module images and two application container images in `module-1`.

Print the list of files in the module image

```
crane export <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>:<MODULE_TAG> - | tar -tf -
```

An example:

```
crane export registry.example.io/modules-source/module-1:v1.23.1 - | tar -tf -
```

The output will be quite large.

Print the list of images of the module's application containers

```
crane export <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>:<MODULE_TAG> - | tar -Oxf - images_digests.json
```

An example:

```
$ crane export registry.example.io/modules-source/module-1:v1.23.1 -  | tar -Oxf - images_digests.json
{
  "backend": "sha256:fcb04a7fed2c2f8def941e34c0094f4f6973ea6012ccfe2deadb9a1032c1e4fb",
  "frontend": "sha256:f31f4b7da5faa5e320d3aad809563c6f5fcaa97b571fffa5c9cab103327cc0e8"
}
```

Configuring extra images

Modules can include additional images (such as vulnerability databases or other supplementary data) by adding an `extra_images.json` file. This file specifies extra images that need to be manually pushed to the registry and are separate from the main module images.

To view the extra images configuration:

```
crane export <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>:<MODULE_TAG> - | tar -Oxf - extra_images.json
```

An example of `extra_images.json` for neuvector vulnerability database:

```
{
  "scanner": 3
}
```

Important notes:

- Extra images must be manually pushed to the module registry under the `extra/` path.
- Use the `d8 mirror pull --only-extra-images` command to pull only extra images.
- Extra images are stored in the registry as `<module-name>/extra/<image-name>`.

Print the list of releases

```
crane ls <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>/release
```

An example:

```
$ crane ls <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>/release
v1.23.1
v1.23.2
alpha
beta
```

In the example above, there are two releases in the container registry; two release channels, `alpha` and `beta`, are also used:

Print the version in use on the `alpha` release channel

```
crane export <REGISTRY_URL>/<MODULE_SOURCE>/<MODULE_NAME>/release:alpha - | tar -Oxf - version.json
```

An example:

```
$ crane export registry.example.io/modules-source/module-1/release:alpha - | tar -Oxf - version.json
{"version":"v1.23.2"}
```

Additional resources

- Global custom resources
- Module `deckhouse` parameters

Disaster resilience

Disaster resilience is the ability of infrastructure based on Deckhouse Kubernetes Platform (DKP) to remain operational in the event of large-scale failures. It is achieved through distributed deployment, automatic traffic switching, and replication of critical components.

DKP implements two approaches to disaster resilience:

- Geo-distribution, which is a distribution of infrastructure components across different availability zones (Multi-AZ) or regions (Multi-Region). This approach helps reduce the impact of infrastructure failures on application availability. For details, refer to Geo-distribution.
- Geo-reserving, which is based on using multiple independent clusters combined into a multi-cluster system. If one of the clusters becomes unavailable, traffic can be automatically redirected to another. For details, refer to Geo-reserving.

Both approaches require configuring network connectivity between nodes and regions, traffic balancing, and proper storage organization. The choice of an architecture depends on the specifics of applications and infrastructure constraints.

Geo-distribution

Geo-distribution, in the context of disaster resilience in Deckhouse Kubernetes Platform (DKP), is an architectural approach that ensures continuous cluster operation by distributing its components across multiple availability zones (Multi-AZ) or regions (Multi-Region). This approach minimizes the risks of incidents of various scales.



When implementing geo-distribution, it’s important to ensure network connectivity between cluster nodes. This is the responsibility of the platform administrator.

Using multiple availability zones (Multi-AZ)

DKP lets you distribute cluster nodes across availability zones. This approach increases application resilience to infrastructure-wide failures and ensures the applications remain functional even if one of the zones in a data center or cloud becomes unavailable.

An **availability zone (AZ)** is the basic unit of a disaster-resilient architecture. It represents a logical group of one or several data centers located within a few kilometers of each other. An AZ can be viewed as an isolated unit (a virtual data center), even though it's a physically distributed infrastructure. Availability zones are part of regions.

Availability zone features

Availability zones are defined by the following features:

- Sufficient physical separation between data centers within the same AZ (usually ≤10 km) to protect from localized incidents (such as floods or fires), while maintaining low network latency.
- Independent power supply for components within the zone (different substations, diesel generators, etc.).
- Use of different providers and network routes by components within the zone.
- Relatively low latency within the zone.

Distributing nodes across zones

With most cloud providers, zone distribution happens automatically. In non-cloud environments, administrators must manually label nodes to indicate the target zone.

For the node labeling process and how to add nodes to groups, refer to "Node management".

Using multiple regions (Multi-Region)

DKP supports distributing cluster nodes across different regions. This approach allows applications to continue running even in the event of a complete outage of a data center or cloud provider.

A **region** is an independent geographic area that includes several (typically at least three) isolated availability zones (AZs). All zones within a region are united into a single logical infrastructure unit with shared management services. A region is a key element in the global distributed architecture used by cloud providers.

> **i** Using multiple regions (Multi-Region) is usually appropriate only when applications can work independently in each region (without reaching out to others). This is due to the relatively high latency between regions. Typical use cases include hosting CDN cache servers or distributed monitoring nodes.

Region features

Regions are defined by the following features:

- Latency differences within a region and between regions. Within a region, latency between AZs can be around 5 ms. Between regions, it can be between 50 and 200 ms.
- Geographic distance. Regions may be widely dispersed. For example, the distance between `europe-west3` (Frankfurt) and `us-east-1` (Virginia) is around 6500 km.
- Physical safety. Providers try to place regions outside of seismically or politically unstable areas.
- Energy redundancy. Regions are usually connected to multiple national power grids (for example, `me-central-1` (UAE) uses 4 independent power sources).

Distributing nodes across regions

Distributing nodes across regions requires the following:

- Labeling each node to indicate the region it belongs to.
- Providing stable communication channels between regions.

For the node labeling process and how to add nodes to groups, refer to "Node management".

Balancing incoming traffic

In a geo-distributed cluster, traffic balancing can be handled either by the cloud provider or by internal load balancers (such as MetalLB). The specific implementation depends on the infrastructure and must be configured by the administrator. For details on traffic balancing options and configuration, refer to Network.

Storage organization

Storage system organization and configuration in a geo-distributed cluster are handled by the administrator. DKP supports various storage options and the choice depends on the specific project needs. For details on supported storage systems, their features and configuration, refer to Storage.

Geo-reserving

Geo-reserving, in the context of disaster resilience in Deckhouse Kubernetes Platform (DKP), is a method of increasing fault tolerance by combining multiple independent Kubernetes clusters into a single multi-cluster system (multi-cluster) distributed across different geographic locations.

DKP offers several tools to unite multiple clusters into a multi-cluster. This approach ensures automatic routing of both external and internal traffic to another cluster with an available copy of the application, in case it becomes unavailable in the original cluster.

Clusters can be combined using:

- The built-in Istio-based Service Mesh.
- DKP's networking features powered by Cilium.

Enabling multi-cluster functionality requires deploying one DKP cluster per region and then connecting them using a declarative API. It is important that the application can run in parallel across different regions, and that there is a stable network connection between the regions. A properly configured system ensures global disaster resilience for the application, guaranteeing uninterrupted operation even if an entire region or cloud becomes unavailable.

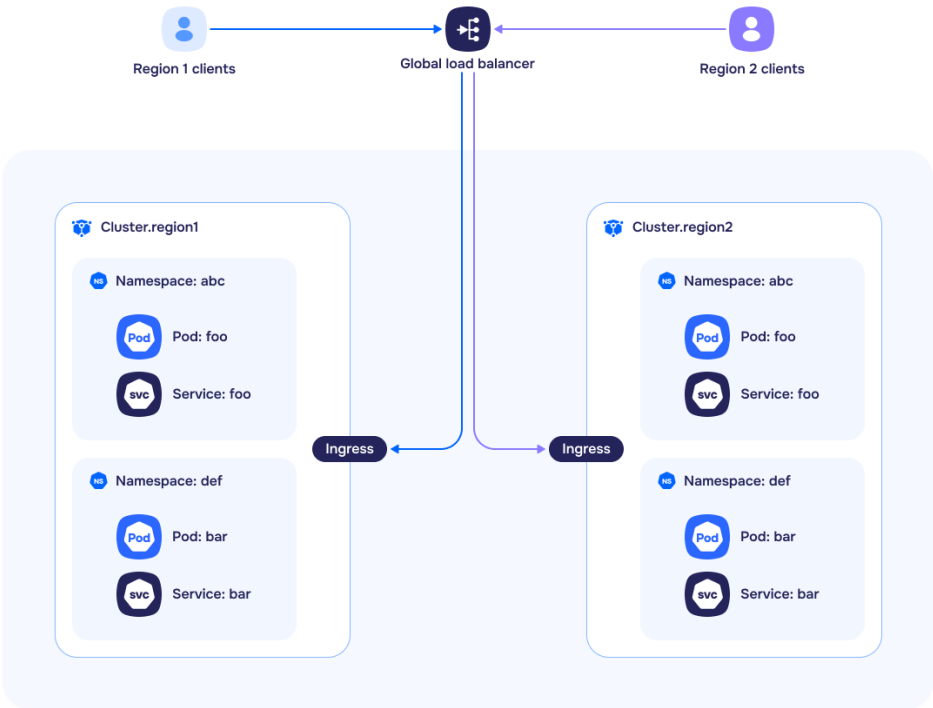
For the process of joining multiple clusters into a multi-cluster, refer to Inter-cluster cooperation.

Balancing incoming traffic in a multi-cluster

To ensure disaster resilience in a multi-cluster, incoming traffic can be balanced by an external load balancer using one of the following methods: `active-active` or `active-standby`. The balancing configuration is handled by the cluster administrator.

Balancing traffic via active-active method

Following the `active-active` method, the external load balancer distributes traffic across multiple active clusters, for example, based on geographic proximity or current workload.

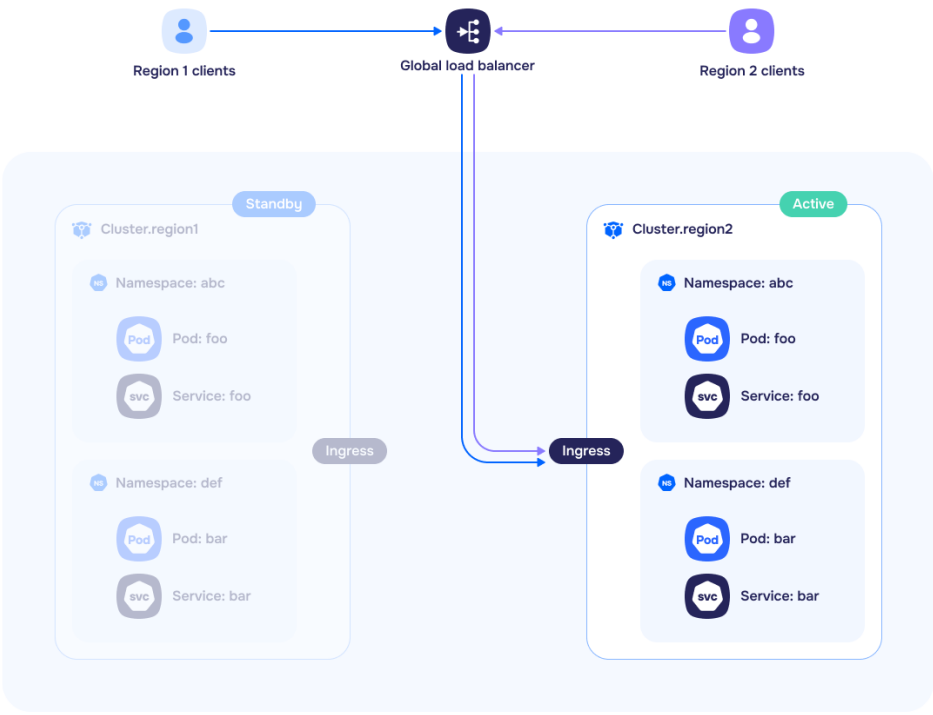


Key aspects of `active-active` balancing:

- All clusters in the multi-cluster process traffic simultaneously.
- Traffic is routed to the nearest and least loaded cluster, or according to other defined rules.
- Shared data is replicated across clusters.
- In the event of a failure in one cluster, traffic is automatically rerouted to the others (automatic failover).

Balancing traffic via active-standby method

Following the `active-standby` method, the external load balancer routes traffic only to the currently active cluster. The standby cluster remains idle and receives traffic only if the primary cluster becomes unavailable.



Key aspects of `active-standby` balancing:

- The active cluster receives all traffic.
- The standby cluster remains idle until the primary cluster fails.
- Traffic failover to the standby cluster is performed automatically upon detection of issues.

Storage organization in a multi-cluster

Storage system organization and configuration in a multi-cluster are handled by the administrator. DKP supports various storage systems. The choice of a suitable solution depends on requirements of performance, fault tolerance, and data synchronization methods. For details on supported storage systems, their features and configuration, refer to Storage.

Updating

Releases

A release is any published version of Deckhouse Kubernetes Platform (DKP). Each release is distributed through release channels with defined delays. DKP publishes two types of releases:

- **Patch version** (for example, from `0.0.1` to `0.0.2`): Includes bug fixes and is published when needed.
- **Minor version** (for example, from `0.0.1` to `0.1.0`): Includes new features and is published every 3–4 weeks.

Release channels

> **i** Up-to-date information about DKP versions available on different release channels is available at releases.deckhouse.io.

DKP uses **five release channels** to gradually roll out new versions. Each new DKP version is first published to the **Alpha** channel and then gradually moves to **Rock Solid**. Updates in less stable channels are made available to a limited number of users, which allows to detect and resolve potential issues before they affect production environments.

- **Rock Solid**: The most stable release channel. Ideal for clusters requiring maximum reliability. Updates are published at least 30 days after the release.

- **Stable**: A stable channel suitable for production clusters. Updates are published at least 14 days after the release.
- **Early Access**: The recommended release channel if you're unsure which one to choose. Suitable for actively evolving clusters (for example, those launching or refining new applications). Updates are published at least 7 days after the release.
- **Beta**: Intended for development clusters, similar to the Alpha channel. Receives versions that have already been tested in Alpha.
- **Alpha**: The least stable channel with the most frequent updates. Targeted at development clusters with a small number of developers. Versions appear immediately after release.

Switching to a more stable channel

When switching to a more stable channel (for example, from `Alpha` to `EarlyAccess`):

1. DKP fetches release data from the `EarlyAccess` channel.
2. It compares this data with existing DeckhouseRelease custom resources in the cluster.
   - If the cluster contains newer releases with `Pending` status (not yet applied), they will be **removed**, since they haven't been published to the new channel.
   - If newer releases have already been marked as `Deployed` (installed successfully), the switch won't take effect immediately. DKP will remain on the current release until a newer version becomes available in the `EarlyAccess` channel.

Switching to a less stable channel

When switching to a less stable channel (for example, from `EarlyAccess` to `Alpha`):

1. DKP fetches release data from the `Alpha` channel.
2. It compares this data with existing DeckhouseRelease custom resources.
3. It applies the update according to the configured update parameters.

Control plane updates

In DKP, the control plane update process is highly automated and safe for both single-master and multi-master clusters. While brief interruptions in API server availability may occur, they do not affect the operation of applications running in the cluster. In most cases, no additional maintenance window is required.

DKP supports the latest five minor Kubernetes versions. A full list of supported versions is available in the corresponding table.

Patch version updates

Patch updates to control plane components (within the same minor version, for example, from `1.27.3` to `1.27.5`) are applied automatically together with DKP updates. Users cannot manage patch updates manually. The process is fully automated by the platform.

Automatic minor version updates

To automatically update the control plane to a new minor version (for example, from `1.28.*` to `1.30.*`), specify `kubernetesVersion: Automatic` in the ClusterConfiguration resource. DKP will select the default Kubernetes version at the time of the update.

Manual minor version updates

To manually update the control plane to a new minor version (for example, from `1.28.*` to `1.30.*`), specify the target version in the `kubernetesVersion` parameter of the ClusterConfiguration resource. For example, `kubernetesVersion: 1.30`.

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

This command initiates an upgrade to the default minor Kubernetes version used by DKP at the time. To track the upgrade progress, check the Kubernetes version in the output of the node description command:

```
d8 k get nodes
```

Process of control plane component updates

1. The control-plane-manager module modifies the manifests of core components (`apiserver`, `controller-manager`, `scheduler`, etc.).
2. The updated components are deployed on all master nodes.

When performing a minor version upgrade:

- If bumping for more than one minor version (for example, from `1.28` to `1.30`), the update happens in stages: `1.28` → `1.29` → `1.30`.
- At each step, the control plane is updated first, followed by the kubelet upgrade on the nodes.

When performing a downgrade:

- Only a single minor version downgrade is supported from the highest version ever used in the cluster.
- The downgrade occurs in reverse order. Kubelet components on the nodes are downgraded first, followed by the control plane components.

After the control plane has been updated, node updates begin:

1. The Bashible API Server updates the kubelet version in the scripts for all NodeGroups.
2. Nodes in different NodeGroups are updated in parallel, but within each NodeGroup, they are updated sequentially (one by one).
3. Each NodeGroup's update settings are taken into account:
   - Manual or automatic update mode.
   - Update windows.
   - Whether Pods should be evicted before updating, etc.
4. One or more candidate nodes that meet all update conditions are selected and updated.
5. The process repeats until all nodes in all groups are updated.
6. When all nodes are updated, the update process is complete.

Retrieving the changelog

Each new version of DKP includes a *changelog*, which is a detailed list of changes, including new features, bug fixes, component updates, and important compatibility notes.

You can find the changelog for a specific DKP version in the Deckhouse release list on GitHub.

A summary of key changes, component version updates, and which cluster components will be restarted is included in the description of the zero patch release: example for DKP v1.68.

Changelog contents

The changelog includes four sections:

- **Know before update**: Critical information to consider before updating. Includes compatibility requirements, pre-update actions, and possible cluster impacts.
- **Features**: Highlights of key improvements and new features introduced in the release.
- **Fixes**: Minor changes, security updates, and performance improvements.
- **Chore**: Technical updates, such as dependency updates, refactoring, build and test pipeline changes, and vulnerability fixes.

Minor versions and zero patch releases

All major changes are listed in the **zero patch release** (for example, `v1.68.0` for the DKP `v1.68`). Before updating to a new minor version (for example, `v1.68`):

1. Review the changelog for the corresponding version.
2. Check if the changes affect your infrastructure.
3. Adjust the cluster configuration if needed.

Checking dependencies before update

Before applying a new release, DKP checks the cluster for potential issues. If any of the following incompatibilities are detected, the update is aborted:

- Unsupported Kubernetes version.
- When `kubernetesVersion: Automatic` is enabled, the update is aborted if:
  - The release introduces a new default Kubernetes version;
  - Monitoring is enabled;
  - The cluster contains resources that are deprecated in the new version.
- The installed Ingress controller version is incompatible with the new release.
- Nodes are running outdated or unsupported operating systems.
- The cluster has an enabled module that is `deprecated` or has been removed in the new release.

Additional resources

- Global custom resources
- Module `control-plane-manager` documentation

Deckhouse subsystem

This subsection describes the architecture of the modules that are part of the Deckhouse subsystem of Deckhouse Kubernetes Platform (DKP).

The Deckhouse subsystem includes the following modules:

- `deckhouse`: Deckhouse controller.
- `console`: Deckhouse web UI.
- `deckhouse-tools`: Provides a web interface for downloading the `d8` CLI utility.

- `documentation`: Provides a web interface with documentation corresponding to the running DKP version.
- `registry`: Manages the configuration of DKP components responsible for working with the container registry and provides an internal image storage.

Web interface modules

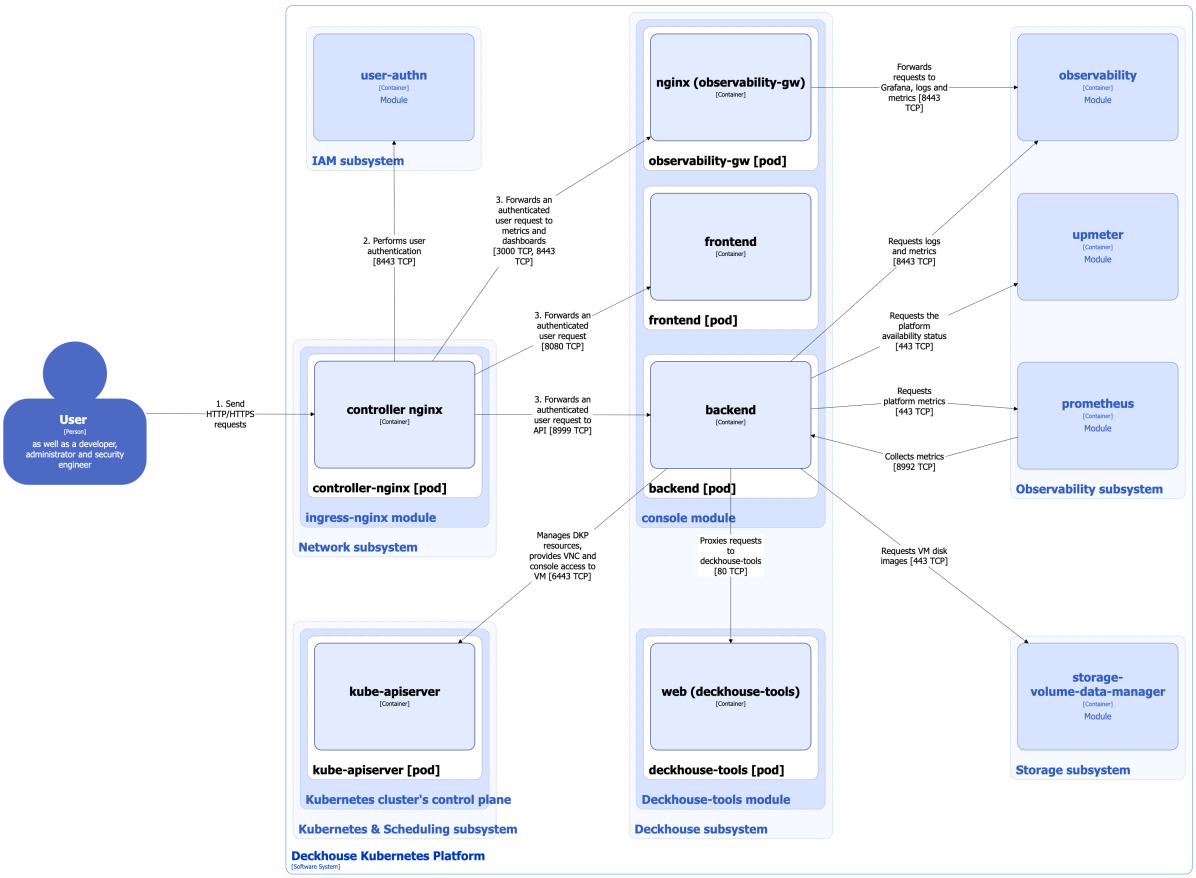
Console module

The `console` module implements the web interface of Deckhouse Kubernetes Platform (DKP), simplifying platform management and enabling the system state monitoring.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `console` module and its interaction with other DKP components are shown in the following diagram:



> **i** The numbers in the diagram indicate the order in which the user's request passes through the `frontend`, `backend`, and `nginx` components.
>
> At steps 1, 2, and 3, the request passes through the Ingress NGINX Controller, where mandatory user authentication is performed by the `user-authn` module. For more information about the architecture of the `user-authn` module, refer to the corresponding documentation section.

Module components

The module consists of the following components:

1. **Frontend**: Consists of a single **frontend** container and provides the web interface for DKP users and

administrators.

2. **Backend**: Consists of a single **backend** container and implements an API interface that provides the following capabilities:
   - Retrieving, creating, deleting, and modifying DKP resources according to the user's permissions.
   - Generating a kubeconfig with the user's profile.
   - Determining the environment in which the DKP cluster is deployed.
   - Detecting DKP and Kubernetes versions.
   - Loading metrics and logs.
   - Loading platform availability information.
   - Downloading VM disks (when the `virtualization` and `storage-volume-data-manager` modules are enabled).
3. **Observability-gw**: Consists of a single **nginx** container and proxies requests to Grafana to embed dashboards into the platform's main web UI, as well as to work with metrics and logs for the selected project.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Establishing VM connections via console and VNC.
   - Creating, deleting, modifying, and tracking DKP resources.
2. **Upmeter**: Retrieves DKP availability information.
3. **Deckhouse-tools**: Forwards requests to download the Deckhouse CLI utility.
4. **Prometheus**: Retrieves system metrics for the platform, such as CPU and RAM usage.
5. **Observability**: Retrieves metrics and logs for the selected project.
6. **Storage-volume-data-manager**: Exports VM disk images.

The following external component interacts with the module:

- **Controller nginx**: Forwards external user requests to the module web interface.

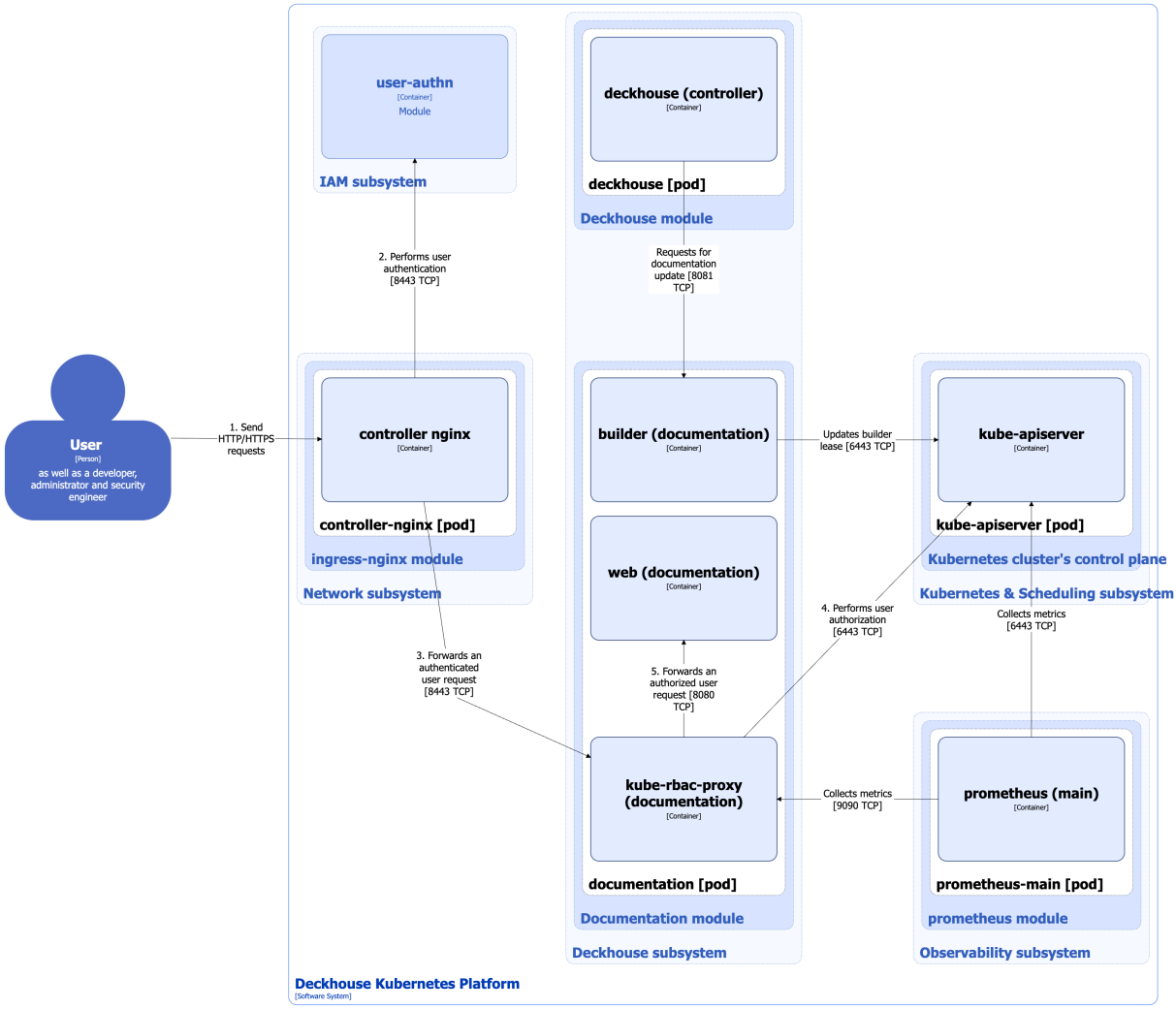
Documentation module

The `documentation` module provides a web interface for documentation corresponding to the running version of DKP.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `documentation` module and its interaction with other DKP components are shown in the following diagram:



> **i** The numbers in the diagram indicate the order in which the user's request passes through the `web` component:
>
> - At steps 1, 2, and 3, the request passes through the Ingress NGINX Controller, where mandatory user authentication is performed by the `user-authn` module.
> - At steps 4 and 5, user authorization based on the Kubernetes RBAC is performed to provide secure access.

Module components

The module consists of the following component:

- **Documentation**: A component that provides the documentation web interface.

  It consists of the following containers:

  - **web**: Main container.
  - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the main container. It is an open source project.
  - **builder**: Sidecar container that dynamically extends the documentation when new DKP modules are installed. The Hugo static site generator is used to render and generate the up-to-date site content.

    The **builder** container automatically creates and updates a Kubernetes Lease resource, placing an endpoint for documentation updates. This endpoint is used by the `deckhouse` module controller to initiate documentation updates when modules are updated or installed. This ensures that changes are promptly rendered in the web interface.

Module interactions

The module interacts with the following components:

- **Kube-apiserver**:
  - Creates and updates the Lease resource.

- Authorizes requests to the documentation web interface.

The following external components interact with the module:

1. **Deckhouse**: Sends requests to update the documentation when the set of modules changes.
2. **Prometheus**: Collects module metrics.
3. **Controller nginx**: Forwards external user requests to the module web interface.

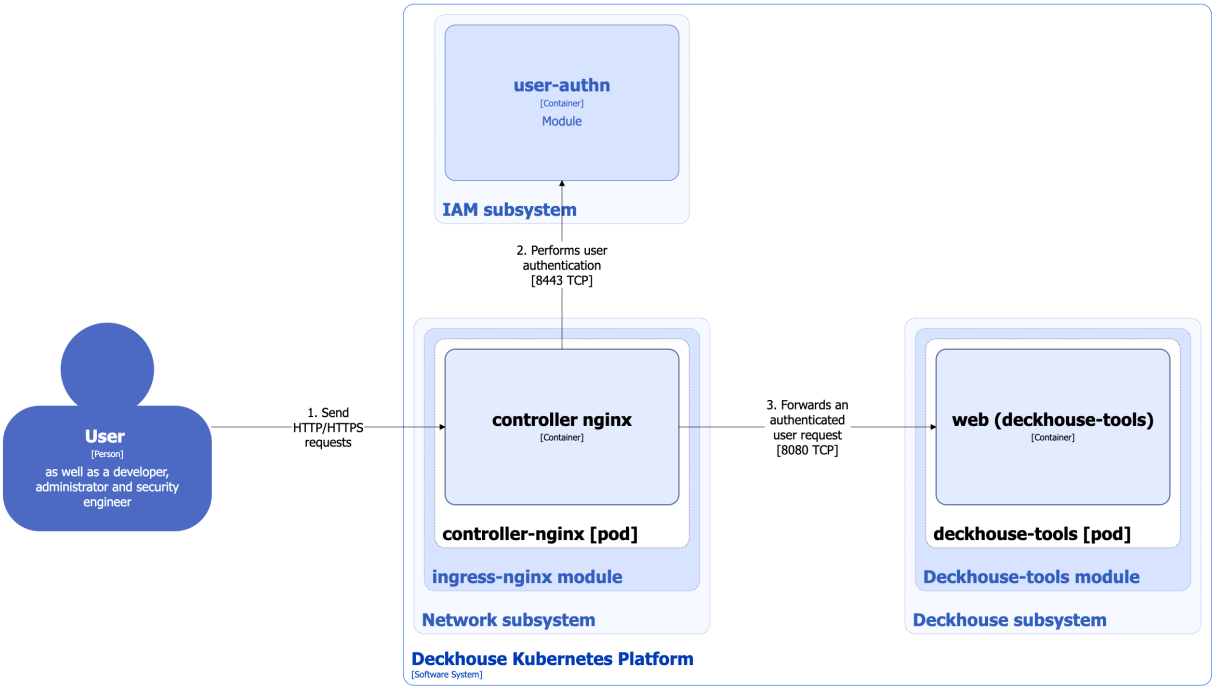
Deckhouse-tools module

The `deckhouse-tools` module provides a web interface with links to download the Deckhouse CLI utility for various operating systems.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `deckhouse-tools` module and its interaction with other DKP components are shown in the following diagram:



> The numbers in the diagram indicate the order in which the user's request passes through the `web` component.
>
> At steps 1, 2, and 3, the request passes through the Ingress NGINX Controller, where mandatory user authentication is performed by the `user-authn` module.

Module components

The module consists of the following component:

- **Deckhouse-tools**: Consists of a single **web** container and provides a web interface with links to download the Deckhouse CLI utility.

Module interactions

The following external component interacts with the module:

- **Controller nginx**: Forwards external user requests to the module web interface.

Additional resources

- Module `console` documentation
- Module `deckhouse-tools` documentation
- Module `prometheus` documentation
- Module `storage-volume-data-manager` documentation
- Module `upmeter` documentation
- Module `virtualization` documentation

Deckhouse module

The `deckhouse` module implements the core of Deckhouse Kubernetes Platform (DKP), performing the following operations:

- Platform updates
- Module configuration management
- Module installation and updates
- Module documentation build triggering
- Validation of custom resources managed by DKP modules

The module manages the following custom resources in the `deckhouse.io` API group:

- Module management:
  - Module: Description, status, and publication of module information.
  - ModuleConfig: Description of user-defined module settings.
  - ModulePullOverride: Description of module version selection overrides.
  - ModuleRelease: Description, publication, and tracking of module releases.
  - ModuleSettingsDefinition: Schema, versions, and transformation rules for module settings.
  - ModuleSource: Description of a module source, repository, or storage.
  - ModuleUpdatePolicy: Rules for module updates and version transition automation.
- Platform management:
  - DeckhouseRelease: An object that defines the DKP release (version) and platform update policy.
- Package management (Marketplace):
  - Application: Description and desired state of an application package (a group of components or an application).
  - ApplicationPackage: Package metadata, sources, and settings.
  - ApplicationPackageVersion: Description of a specific package version and its parameters.
  - PackageRepository: An object that describes a package repository source and its parameters.
  - PackageRepositoryOperation: Operations on package repositories, such as synchronization or updates.
- Utility management:
  - CNIMigration: Container Network Interface (CNI) migration process, including migration parameters and status.
  - CNINodeMigration: Status and management of CNI migration at the individual node level.
  - ObjectKeeper: A resource that links Kubernetes resources using `ownerReference`.
  - ModuleDocumentation: Description of parameters for generating and storing module documentation.
- Management of custom resources controlled by DKP modules:
  - ConversionWebhook: Settings and handlers for resource conversion webhooks.
  - ValidationWebhook: Settings and handlers for resource validation webhooks.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `deckhouse` module and its interaction with other DKP components are shown in the following diagram:

Module components

The module consists of the following components:

1. **Deckhouse** (Deployment): A controller that implements platform management operations.

   The controller coordinates platform management tasks using the queueing mechanism.

   The Deckhouse controller can run in standard mode or in hook isolation mode. To enable it, create the

`chroot-mode` ConfigMap in the `d8-system` namespace. In isolation mode, shell hooks and module enable scripts run in a chroot environment with a limited set of mounted directories, isolating them from the controller container file system.

If High Availability (HA) mode is enabled, multiple Deckhouse controller instances are started. To ensure correct behavior, Deckhouse controllers perform leader election using the `deckhouse-leader-election` Lease resource. The controller elected as leader performs all platform management operations.

In addition, you can set the following parameters using the Deckhouse controller:

Module configuration parameter	Description
`logLevel`	Logging level
`bundle`	Set of modules enabled by default
`releaseChannel`	Release channel
`update.mode`	Update mode
`update.windows.days`	Update windows

For details about module settings, refer to the module documentation section.

It consists of the following containers:

- **init-downloaded-modules**: Init container that prepares the directory structure required for module operations.
- **deckhouse**: Main container.
- **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the main container component debug interface.

2. **Webhook-handler** (Deployment): Consists of a single **handler** container and implements a generic webhook for conversion and validation of custom resources managed by DKP.

   The component watches ConversionWebhook and ValidationWebhook custom resources and, based on them, generates hook Python files for shell-operator from templates. When `kube-apiserver` sends resource validation or conversion requests, shell-operator runs the required hook and returns the processing result.

3. **Cni-migration-manager** (Deployment): An optional component running on control plane nodes, consisting of a single **manager** container. The component manages the network plugin (CNI) switching process in the DKP cluster and records the current state in the CNIMigration custom resource. Migration to Flannel, Simple bridge, and Cilium is supported. For details on switching CNI in the cluster, refer to the corresponding guide.

   > **i** The component is created by the `detect-cni-migration` global hook when the CNIMigration custom resource exists. The CNIMigration resource is created manually by an administrator or by running the `d8 network cni-migration switch --to-cni <target cni>` command.

4. **Cni-migration-agent** (DaemonSet): An optional component running on all cluster nodes, consisting of a single **agent** container. The component watches the CNIMigration custom resource and manages the CNINodeMigration custom resource that reflects migration state for a specific node.

   > **i** The component is created by the `detect-cni-migration` global hook when the CNIMigration custom resource exists. The CNIMigration resource is created manually by an administrator or by running the `d8 network cni-migration switch --to-cni <target cni>` command.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Working with custom resources in the `deckhouse.io` API group.
   - Watching Pod and DaemonSet resources, as well as restarting Pods when the network plugin is changed.
   - Watching resources described in the ObjectKeeper custom resource.
   - Creating and updating Lease resources.
   - Creating, deleting, modifying, and watching resources described in DKP modules.
   - Authorizing requests.
2. **Documentation**: Updating documentation when a DKP module is added or updated.
3. **Image registry**: Retrieving module component images along with metadata when the `registry` module is installed in `Unmanaged` mode.

4. **`registry` module**: Retrieving module component images along with metadata when the `registry` module is installed in one of the following modes: `Direct`, `Proxy`, or `Local`.

The module is interacted with by the following external components:

- **Kube-apiserver**: Validation and conversion of DKP custom resources.
- **Prometheus-main**: Collecting metrics from the `deckhouse` and `webhook-handler` containers.

Additional resources

- Global custom resources
- Module `deckhouse` documentation
- Module `documentation` documentation
- Module `registry` documentation

Queuing mechanism

The `deckhouse` module implements the core of Deckhouse Kubernetes Platform (DKP), performing various platform management operations using a queueing mechanism. For more information about the module architecture, refer to the corresponding documentation section.

The Deckhouse controller implements addon-operator and marketplace queues.

Addon-operator queues

The **addon-operator queues** are the primary processing mechanism for built-in and external DKP modules. The queue is implemented in shell-operator and extended with addon-operator task types. The Deckhouse controller synchronizes ModuleConfig custom resources and updates global or module values for addon-operator.

Task types:

Task	Function
GlobalHookRun	Runs global hooks (`onStartup`, `beforeAll`, `afterAll`, `kubernetes`, `schedule`)
GlobalHookEnableKubernetesBindings	Enables Kubernetes monitors from global hooks
GlobalHookWaitKubernetesSynchronization	Blocks the queue until global hooks with `executeHookOnSynchronization: true` complete
GlobalHookEnableScheduleBindings	Registers cron tasks in the addon-operator scheduler
DiscoverHelmReleases	Finds “extra” Helm releases after the first `converge`
ApplyKubeConfigValues	Applies changes from ModuleConfig
ConvergeModules	Full `converge` cycle for all modules
ModuleRun	Configures or updates a module, including subtasks: `onStartup` -> `sync` -> `beforeHelm` -> `helm` -> `afterHelm`
ParallelModuleRun	Launches modules in batch and in parallel
ModuleDelete	Removes a module (`helm delete`, `afterDeleteHelm`)
ModuleHookRun	Runs a module hook on event
ModuleEnsureCRDs	Installs module CRDs
ModulePurge	Removes unknown Helm release

Addon-operator queue types:

Queue	Name	Purpose
Main	`main`	Runs global hooks at startup, installs critical modules, configures and deletes modules
Parallel	`parallel_queue_0` ... `parallel_queue_19`	Parallel ModuleRun with module dependencies taken into account (20 queues)
Hook queues	From task config (hook)	Tasks for a specific module and global hooks

Each queue is a separate pipeline with one worker. The queue has the following properties:

- Tasks can be inserted both at the tail (`AddLast`) and at the head (`AddFirst`) of the queue.
- Tasks are executed from the head of the queue.
- Multiphase operations are supported with the following operations:
  - `AddHeadTasks` : Insert subtasks before the current task.
  - `AddTailTasks` : Insert a subtask at the end of the queue after the current task succeeds.
  - `AddAfterTasks` : Insert a subtask right after the current task.
- A task is executed until success unless `allowFailure: true` is set in task parameters.
- On error, exponential restart (backoff) is applied, starting with a 5-second delay between retries.

> ⚠ If a task cannot complete successfully and `allowFailure: true` is not set in its parameters, that task blocks the queue it runs in. Tasks in different processing queues do not block each other.

At startup, the Deckhouse controller creates `main` and `parallel_queue_0..19` queues and adds the following tasks to `main` in this order:

- GlobalHookRun (onStartup): For each global hook.
- GlobalHookEnableScheduleBindings: To enable cron scheduling.
- GlobalHookEnableKubernetesBindings: To enable global tasks that watch Kubernetes resources.
- GlobalHookWaitKubernetesSynchronization.
- ConvergeModules (OperatorStartup): First converge of all modules.

After ConvergeModules, the controller adds the DiscoverHelmReleases task to clean up unknown Helm releases.

The module processing order in the ConvergeModules task is determined by several attributes:

- Module criticality: The `critical` parameter in the module `module.yaml` configuration.
- Module *weight*: Numeric module processing order. The higher the number, the later the module is processed. The weight is taken from the `weight` parameter in module `module.yaml`; if missing or set to 0, the default weight 900 is used. If no module config file exists, the weight is taken from the numeric prefix of the module directory name (for example, `040-node-manager` means weight 40). If the weight cannot be obtained from the directory name, weight 100 is used.
- Module dependencies: A list of modules that must be installed before the current module.

Based on these attributes, the Deckhouse controller scheduler defines processing order according to the following principles:

- For critical modules, module weight is considered in ascending order and tasks are put into the `main` queue.
- For non-critical modules, module weight is not considered and tasks are put into `parallel_queue_0..19`.
- For all modules, dependencies on other modules are considered.

If critical modules can be processed in parallel, the Deckhouse controller scheduler places the ParallelModuleRun task into the `main` queue with the list of modules. The ParallelModuleRun task starts a ModuleRun task for each module in `parallel_queue_0..19` and waits for completion, thereby blocking the `main` queue. If an error occurs while processing a ModuleRun task, the scheduler moves this task to the end of the queue and starts the next queue task.

The process of installing critical modules is shown in the following diagram:

The process of installing non-critical (functional) modules is shown in the following diagram:

If more than one identical task is added to a queue, all duplicates are removed when such a task starts, as part of deduplication.

To view addon-operator queues, use the `d8 system queue list` command.

Marketplace queues

The **Marketplace queues** are a queue implementation used by Marketplace functionality.

Each queue served by the Deckhouse controller for Marketplace has the following properties:

- FIFO (First In First Out): Defines strict task execution order. The first task in queue is executed first.
- Strict sequential execution of tasks, one task at a time.
- Tasks are started only on events (event-driven), without polling any resources.
- Restart on errors with exponential backoff between retries, starting at 15 seconds and capped at 1 minute, with no limit on retry attempts.
- Supports cascading task cancellation on version changes or package deletion.

Queue types:

Name	Purpose
`{packageName}`	Lifecycle: Deploy, Load, Configure, Enable, Run, Disable, Undeploy
`{packageName}/{hookQueue}`	Running hooks triggered by Kubernetes or schedule events (queue from hook binding)
`{packageName}/{hookQueue}/sync`	Hook synchronization at startup (WaitForSynchronization)

Task types:

Task	Function
Deploy	Downloads or mounts the package image
Load	Parses configuration, creates Application or Module, registers in scheduler
Configure	Applies Application or Module settings using the parameter store
Enable	Enables hooks, performs parameter synchronization, runs OnStartup hook
Run	Runs subtasks when installing Application or Module: `BeforeHelm` -> `helm Upgrade` -> `AfterHelm`
HookRun	Runs a hook on event
HookSync	Performs initial synchronization of Kubernetes binding
Disable	Removes Helm, disables hooks, clears hook queues
Undeploy	Removes the package from disk

Task execution in one queue does not block task execution in another queue.

To view Marketplace queues, run the following command:

⚠ When performing module installation or configuration tasks in addon-operator, the Deckhouse controller pauses Marketplace queue processing.

Additional resources

- Global custom resources
- Module `deckhouse` documentation

Registry module

The `registry` module manages the registry settings for Deckhouse Kubernetes Platform (DKP) components.

The module can operate in the following modes:

- `Direct`: Provides direct access to an external registry via the fixed address `registry.d8-system.svc:5001/system/deckhouse`. This fixed address prevents Deckhouse images from being re-downloaded and components from being restarted when registry parameters are changed. The fixed address is replaced by the actual address of the external registry when sending requests by configuring `mirroring/rewrite` section in the configuration of the containerd component and by using the internal registry-incluster-proxy component that proxies requests to the external registry.
- `Proxy`: Using an internal caching proxy registry that accesses an external registry, with the caching proxy registry running on control-plane (master) nodes. This mode reduces the number of requests to the external registry by caching images. Cached data is stored on the control-plane (master) nodes. Access to the internal registry is via the fixed address `registry.d8-system.svc:5001/system/deckhouse`, similar to the `Direct` mode.
- `Local`: Using a local internal registry, with the registry running on control-plane (master) nodes. This mode allows the cluster to operate in an isolated environment. All data is stored on the control-plane (master) nodes. Access to the internal registry is via the fixed address `registry.d8-system.svc:5001/system/deckhouse`, similar to the `Direct` and `Proxy` modes.
- `Unmanaged`: Operation without using the internal registry. Access within the cluster is performed directly to the external registry.

For more details about the configuration and usage examples of the module, refer to the module

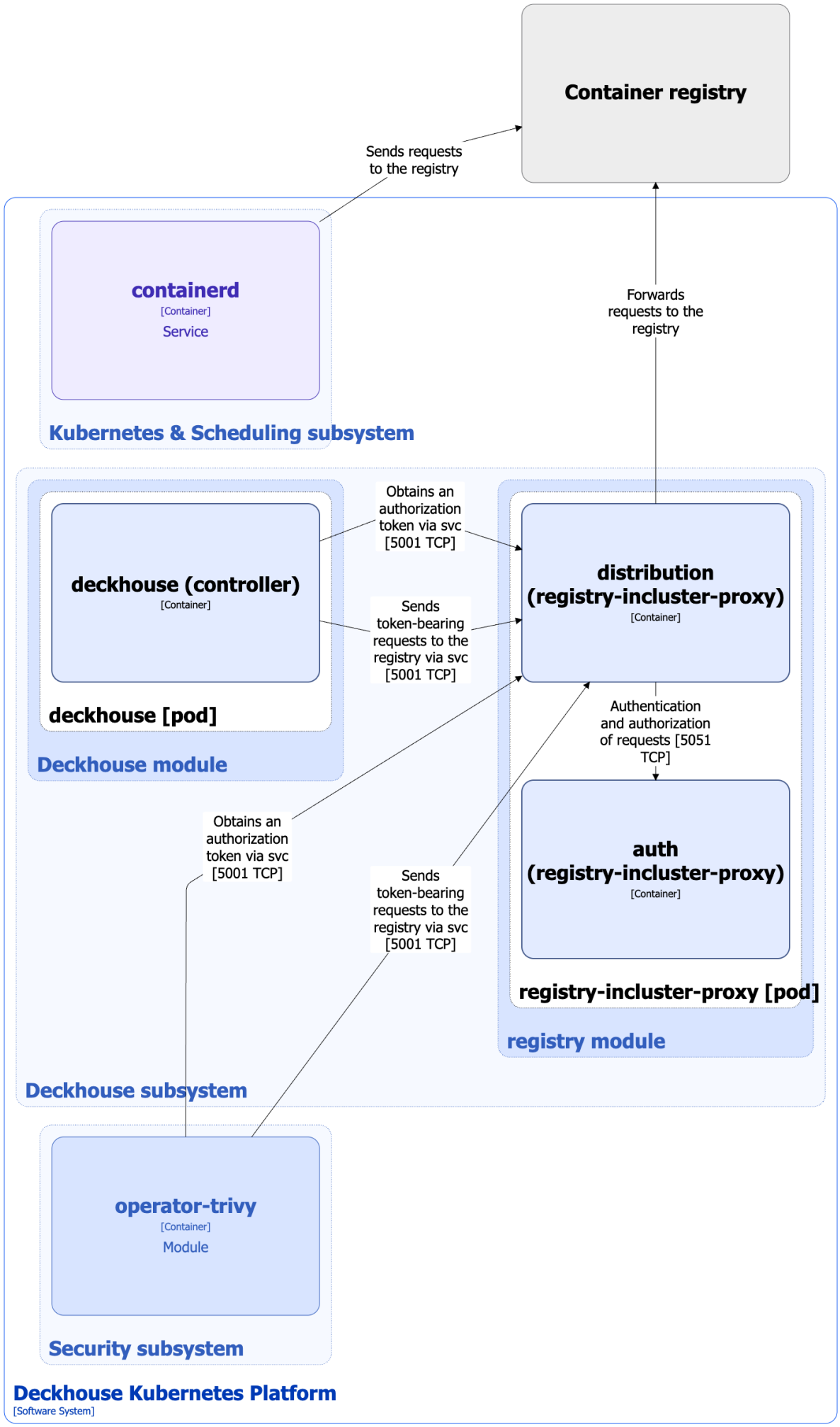
documentation.

Module architecture

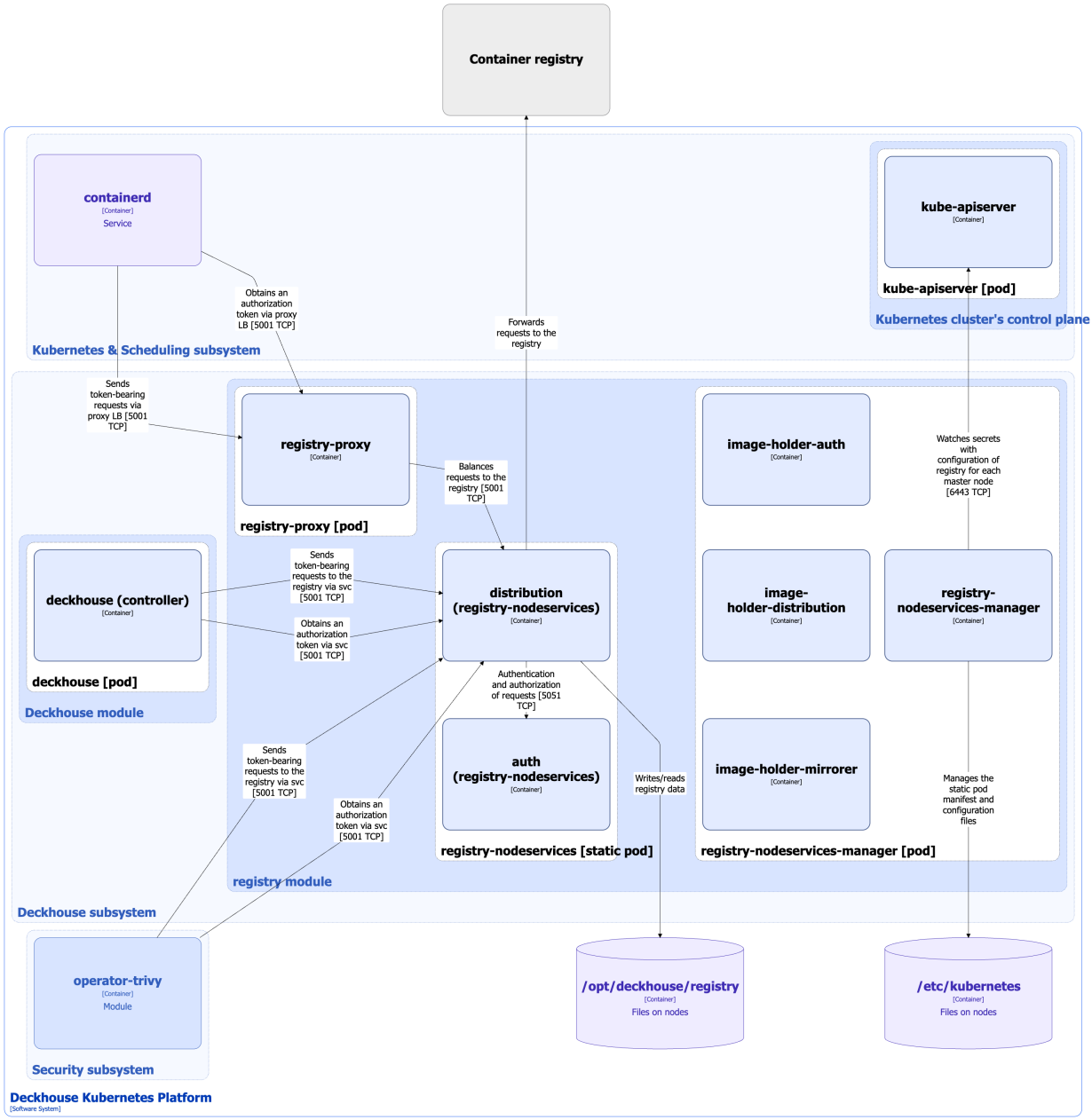
> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `registry` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:

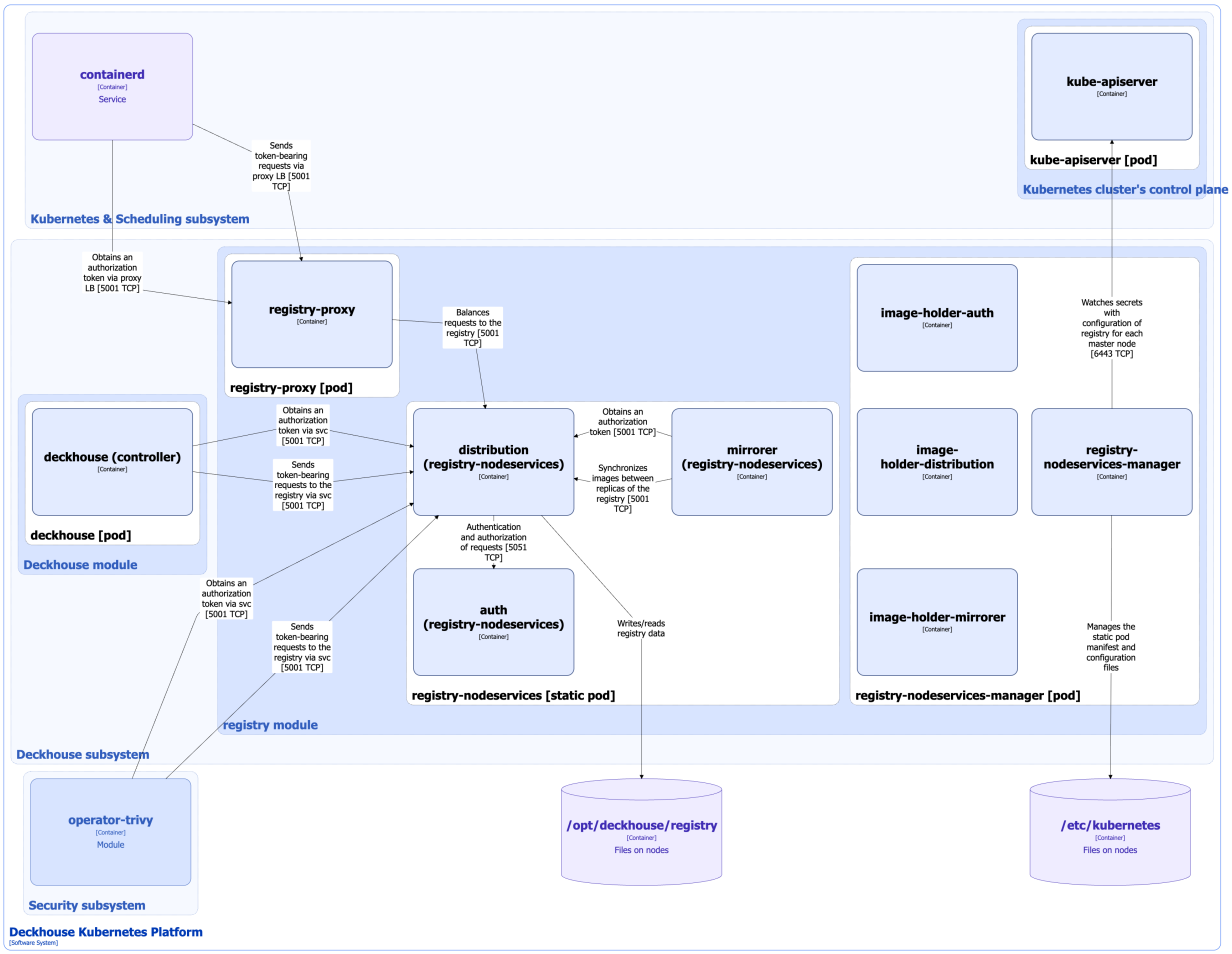
The `registry` module in `Direct` mode:


Container registry
Sends requests to the registry
Forwards requests to the registry
containerd
[Container]
Service
Kubernetes & Scheduling subsystem
deckhouse (controller)
[Container]
deckhouse [pod]
Deckhouse module
Obtains an authorization token via svc [5001 TCP]
Sends token-bearing requests to the registry via svc [5001 TCP]
distribution (registry-incluster-proxy)
[Container]
Authentication and authorization of requests [5051 TCP]
auth (registry-incluster-proxy)
[Container]
registry-incluster-proxy [pod]
registry module
Obtains an authorization token via svc [5001 TCP]
Sends token-bearing requests to the registry via svc [5001 TCP]
Deckhouse subsystem
operator-trivy
[Container]
Module
Security subsystem
Deckhouse Kubernetes Platform
[Software System]

The `registry` module in `Proxy` mode:



The `registry` module in `Local` mode:



Module components

The module consists of the following components:

1. **Registry-incluster-proxy** (`Direct` mode): A container registry based on Distribution. Distribution is an open-source project that provides a framework for storing and distributing container images and other content using the OCI Distribution Specification. Registry-inclusion-proxy is installed in the cluster as a Deployment, does not store or cache images, but forwards requests to the external registry specified in Direct mode settings in `deckhouse` ModuleConfig.

   Registry-incluster-proxy consists of the following containers:

   - **distribution**: Main container.
   - **auth**: Sidecar container that implements authentication and authorization when accessing the registry. It is an open-source project.

     The principle of operation:

     1. **Request to the authentication service**. When the client tries to access the registry, distribution returns a `401 Unauthorized` HTTP response with the `WWW-Authenticate` header indicating how to authenticate.
     2. **Getting a token**. The client sends a request to the authentication service via the distribution service (for example, `registry.d8-system.svc:5051/auth`) using the values `service` and `scope` from the header `WWW-Authenticate`. The auth service returns an opaque Bearer token that represents authorized client access.
     3. **Token usage**. After receiving the token, the client resends the original request to distribution, including the token in the `Authorization` header.
     4. **Access verification**. Distribution validates the token and the `claims` contained in it, and then begins the image download session.

2. **Registry-nodeservices** (`Proxy` or `Local` modes): A container registry based on Distribution running on control plane (master) nodes. The registry is launched as a static pod, and its lifecycle is managed by the registry-nodeservices-manager component. The data is stored in the directory `/opt/deckhouse/registry` on the control plane (master) nodes.

   Registry-nodeservices consists of the following containers:

   - **distribution**: Main container.

    - **auth**: Sidecar container for authentication and authorization (described above).
    - **mirrorer** (`Local` mode): Sidecar container that synchronizes images between distribution replicas running on control plane (master) nodes.

3. **Registry-proxy** (`Proxy` or `Local` modes): A load balancer installed on each cluster node that provides high availability for registry. The CRI accesses the registry via the load balancer. The settings to access the load balancer are specified in the containerd configuration.

   It consists of a single container:

    - **registry-proxy**: A container built based on a standard NGINX image. The registry-proxy-reloader process is also running in the container and restarts the NGINX load balancer processes when its configuration changes.

4. **Registry-nodeservices-manager** (`Proxy` or `Local` modes): A controller that manages the lifecycle of registry-nodeservices components. Registry-nodeservices-manager performs the following operations:

    - Renders the manifest of the registry-nodeservices static pod from templates and saves it in the `/etc/kubernetes` directory on the control plane (master) nodes. Kubelet detects created manifests and runs registry-nodeservices static pod.
    - Saves configuration files necessary for the operation of registry-nodeservices components in the `/etc/kubernetes/registry` directory on control plane (master) nodes.
    - Deletes the static pod manifest in the `/etc/kubernetes` directory and configuration files in the `/etc/kubernetes/registry` directory, thereby deleting the local container registry, in case the module switches to the `Direct` or `Unmanaged` mode.

   Registry-nodeservices-manager consists of the following containers:

    - **registry-nodeservices-manager**: Main container.

    - A set of sidecar containers used to pre-pull images of registry-nodeservices components. These containers remain paused and serve only as image holders:

        - **image-holder-auth**
        - **image-holder-distribution**
        - **image-holder-mirrorer**

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

    - Checks that the current node is the master.
    - Checks that all containers of the registry-nodeservices static pod are pulled and status of registry-nodeservices pod is `Ready`.
    - Applies configuration changes from `registry-node-config-<Node Name>` Secret to the registry-nodeservices static pod.

2. **External container registry**: The module forwards internal requests to the external container registry.

The following external components interact with the module:

1. **Containerd**: Sends requests to the registry to pull images used to create containers.
2. **Deckhouse-controller**: Sends requests to the registry to pull images used to install modules.
3. **Operator-trivy (as well as other components that access the registry directly)**: Sends requests to the registry to download images, for example, to perform security tests (in case of operator-trivy).

Additional resources

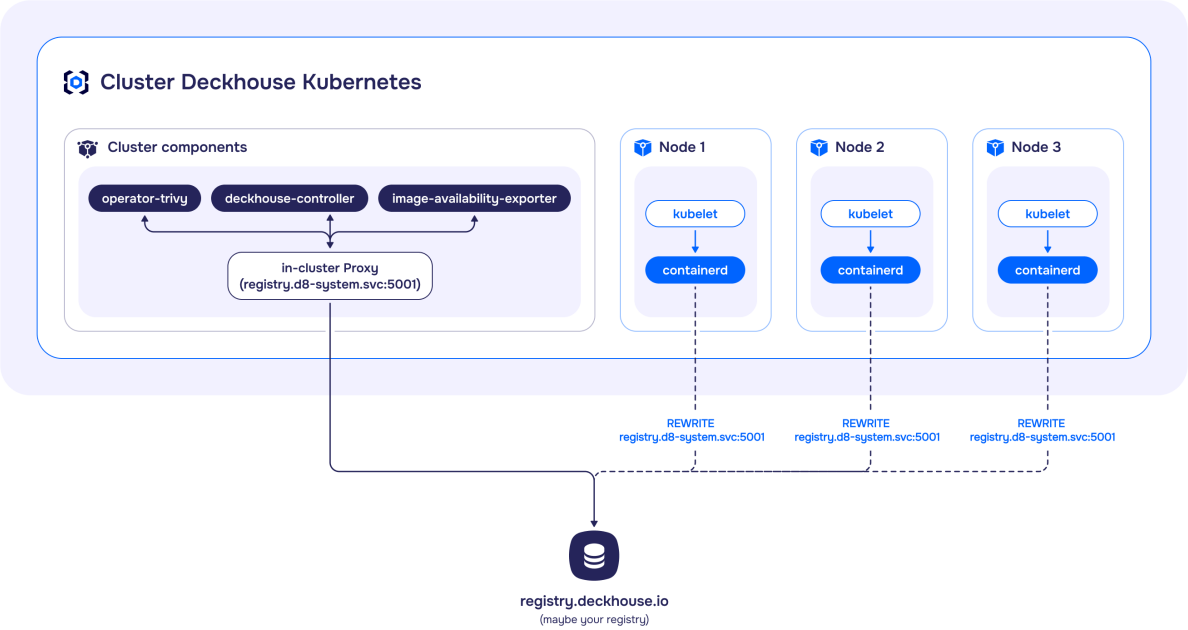
- Module `deckhouse` parameters
- Module `registry` documentation

Direct mode architecture

When the `registry` module is set to the `Direct` mode, container registry requests are processed directly, without intermediate caching.

CRI requests to the registry are redirected based on its configuration, which is defined in the `containerd` configuration.

For components such as `operator-trivy`, `image-availability-exporter`, `deckhouse-controller`, and others that access the registry directly, requests go through the in-cluster proxy located on the master nodes.



For more information about the `Direct` mode, see the section about managing the internal container image registry.

Additional resources

- Module `registry` documentation
- Module `operator-trivy` documentation

Marketplace

Marketplace is the subsystem in Deckhouse Kubernetes Platform (DKP) that manages the lifecycle of delivery units called **Packages**. A Package can be either an **Application** (a user workload deployed into a namespace) or a **Module** (a cluster capability extension). Currently, only Applications are supported; Module support is planned for a future version.

Marketplace is available starting from DKP version 1.76.

Sections

The Concepts section describes the Package abstraction, resource model, Application constraints, and the full scan-to-deploy lifecycle.

The Application development section describes how to create an Application package from scratch: bootstrapping, project structure, `package.yaml`, CI/CD, and artifact layout in the registry.

The Nelm annotations section covers the full set of Nelm annotations used in Application templates to control deployment order, resource lifecycle, tracking, and logging.

The Hooks section describes how to write Go hooks for Applications using the `ApplicationHookInput` type from module-sdk.

Concepts

Package types

A **Package** is an abstract unit that encompasses either an **Application** or a **Module**. The distinction is based on scope and purpose:

Characteristic	Module	Application
Purpose	Infrastructure extension for the cluster	User workload
Scope	Cluster-wide (one per cluster)	Namespaced (unlimited instances)
Multiple instances	No (1:1 with the cluster)	Yes (N instances in different namespaces)
Enabled by default	Can be enabled via bundle	Only by explicit user action
CRD creation	Allowed	Forbidden

Characteristic	Module	Application
Cluster-wide objects	Allowed	Forbidden

Resource model

Deckhouse Kubernetes Platform (DKP) Marketplace uses five custom resources:

```
flowchart TD
    PR[PackageRepository] -->|triggers| PRO[PackageRepositoryOperation]
    PR -->|populates| APV[ApplicationPackageVersion]
    APV -->|aggregated by| AP[ApplicationPackage]
    APV -->|referenced by| APP[Application\nnamespace-scoped]
```

Resource	Short	Scope	Role
`PackageRepository`	—	Cluster	Registry connection and scan schedule
`PackageRepositoryOperation`	`pro`	Cluster	Scan job that discovers versions
`ApplicationPackageVersion`	`apv`	Cluster	One per discovered package version; carries metadata, OpenAPI schemas, and requirements
`ApplicationPackage`	—	Cluster	Informational aggregate: which repos have the package, how many instances use it
`Application`	`app`	Namespace	Installed instance; drives Nelm deployment

ApplicationPackageVersion content

Each ApplicationPackageVersion object carries:

- `status.packageMetadata.description` — localized (`en`/`ru`) package description
- `status.packageMetadata.category` — catalog category
- `status.packageMetadata.stage` — maturity stage (`Preview`, `General Availability`, etc.)
- `status.packageMetadata.requirements` — DKP and Kubernetes version constraints; module dependencies (`mandatory`, `conditional`, `anyOf`, `noneOf`)
- `status.packageMetadata.versionCompatibilityRules` — upgrade and downgrade rules
- `status.packageSchemas.settingsSchema` — OpenAPI v3 schema used to validate `Application.spec.settings`
- `status.packageSchemas.valuesSchema` — OpenAPI v3 schema for effective values passed to hooks and templates

Scan-to-deploy lifecycle

1. Administrator creates PackageRepository.
2. DKP creates a PackageRepositoryOperation automatically (first scan on creation, then every `scanInterval`).
3. The operation scans the registry and creates ApplicationPackageVersion objects for each discovered version.
4. User creates an Application in their namespace referencing `packageName`, `packageVersion`, and optionally `packageRepositoryName`.
5. DKP validates `spec.settings` against the `settingsSchema` from the corresponding ApplicationPackageVersion.
6. Nelm deploys the Helm templates from the package bundle.
7. Conditions on the Application reflect deployment progress: `Installed` → `ConfigurationApplied` → `Scaled` → `Ready`.

Application constraints

All constraints exist to enforce namespace isolation and prevent Applications from interfering with cluster-level resources.

Functional constraints

1. **No CRD creation** — Application templates must not include `CustomResourceDefinition` objects.
2. **No cluster-wide objects** — all resources created by an Application must be namespaced.
3. **No cross-Application dependencies** — an Application can declare dependencies only on Modules (via

`requirements.modules` in `package.yaml`), not on other Applications.
4. **Hooks are namespace-scoped** — hooks must not read or write resources outside their own namespace.
5. **Manual install only** — Applications are never activated by default; installation requires explicit user action.

Naming constraints

Kubernetes limits Pod names to 63 characters. An Application pod name is composed of:

- instance name — ≤24 chars
- resource name — ≤24 chars
- deployment suffix — 15 chars

Therefore:

- **Application instance name** (`metadata.name`): at most **24 characters**
- **Resource name inside the Application** (e.g., Deployment name suffix): at most **24 characters**

Example: instance `redis-cache` (11 chars) + resource `master-deployment` (17 chars) + suffix (15 chars) = 43 chars total — fits within the 63-character limit.

Additional resources

- Global custom resources

Application development

Prerequisites

Install `deckhouse-cli` (`d8`):

```
sh -c "$(curl -fsSL https://raw.githubusercontent.com/deckhouse/deckhouse-cli/main/tools/install.sh)"
```

Log in to the package registry with your license token:

```
d8 dk cr login -u license-token dev-registry.deckhouse.io --password <YOUR_TOKEN>
```

Bootstrapping a new Application

`d8 package bootstrap application <name>` creates a `<name>/` directory in the current working directory with the package skeleton and initializes a git repository with the first commit.

```
d8 package bootstrap application myapp --hooks
cd myapp
git remote add origin <gitlab-repo.git>
git push --set-upstream origin main
```

Flags:

Flag	Description
`--hooks`	Generate a Go hooks skeleton
`--werf`	Use werf for image builds
`--extended`	Add an extended set of files
`-o, --output <path>`	Custom output path (default: `<cwd>/<name>`)

Project structure

```
myapp/
├── .gitignore
├── .gitlab-ci.yml          # CI/CD pipeline
├── changelog.yaml
├── docs/
│   └── README.md           # Application documentation
├── hooks/                  # Go hooks
│   ├── hooks.yaml
│   └── batch/
│       ├── go.mod
│       ├── go.sum
│       ├── main.go
│       └── triggers/
│           └── hook.go
├── images/                 # Image sources or pull instructions
│   └── myapp/
│       └── werf.inc.yaml
├── openapi/
│   ├── config-values.yaml  # OpenAPI schema for Application.spec.settings
│   └── values.yaml         # OpenAPI schema for Helm values
├── oss.yaml
├── package.yaml            # Package manifest
└── templates/              # Helm templates
    ├── deployment.yaml
    ├── registry-secret.yaml
    └── service.yaml
```

package.yaml

The central manifest for an Application package. Defines metadata, type, requirements, and compatibility.

```
apiVersion: v1
type: "Application"
name: redis
descriptions:
  ru: "Redis - in-memory база данных"
  en: "Redis - in-memory database"
# Injected automatically at build time.
version: "v1.0.1"
stage: "Preview"
category: "Databases"
# Environmental Requirements.
requirements:
  deckhouse:
    constraint: ">= 1.70"
  kubernetes:
    constraint: ">= 1.31"
  modules:
    mandatory:
      - name: cert-manager
        constraint: ">= 1.0.0"
```

Field reference:

Field	Required	Description
`name`	Yes	Unique package name
`descriptions`	Yes	Localized description for catalog and UI (`ru`, `en`)
`version`	Yes	Semver version; injected at build time
`type`	Yes	`Application` or `Module`
`stage`	Yes	Maturity stage (`Preview`, `General Availability`, etc.)
`category`	Yes	Category for catalog classification
`requirements.deckhouse`	No	Minimum DKP version constraint
`requirements.kubernetes`	No	Minimum Kubernetes version constraint

Field	Required	Description
`requirements.modules`	No	Module dependencies (semver constraints)

Local build

Build and push the package to a registry:

```
d8 package build -v v0.0.1 -r dev-registry.deckhouse.io/deckhouse/packages
```

For local development, use the payload-registry module as a personal registry.

Linting

Validate the package structure and configuration:

```
d8 package verify
```

Reports errors and warnings based on `.pkglint.yaml` and built-in rules.

CI/CD setup

Environment variables

Variable	Description
`PACKAGES_REGISTRY_LOGIN`	Registry login for publishing
`PACKAGES_REGISTRY_PASSWORD`	Registry password or token

Triggering a release

The pipeline is triggered by a semver git tag:

```
git tag v0.1.0
git push origin v0.1.0
```

The pipeline builds the package and pushes it to the registry. Once the pipeline completes, the package version is available for scanning via PackageRepository.

OCI artifact layout in the registry

```
registry.deckhouse.io/deckhouse/<edition>/packages:<name>
    Package name tag — for listing support

registry.deckhouse.io/deckhouse/<edition>/packages/<name>:<version>
    Bundle — contains templates, openapi/, hooks/

registry.deckhouse.io/deckhouse/<edition>/packages/<name>/extra/<image>:<version>
    Additional images (application containers)

registry.deckhouse.io/deckhouse/<edition>/packages/<name>/version:<version>
    Version metadata — contains package.yaml, version.json, changelog.yaml

registry.deckhouse.io/deckhouse/<edition>/packages/<name>/version:<release-channel>
    Recommended version for a release channel
```

Bundle contents

The main bundle image (`<name>:<version>`) contains:

```
├── package.yaml       # Package manifest
├── openapi/           # Settings and values schemas
├── templates/         # Helm templates
└── hooks/             # Lifecycle hooks
```

Version metadata image contents

The metadata image (`<name>/version:<version>`) contains:

```
├── package.yaml       # Package manifest
├── version.json       # Semver version
└── changelog.yaml     # Release notes
```

Nelm annotations

Application templates are rendered and deployed by **Nelm**. Nelm extends the standard Helm behavior with annotations that control deployment order, resource lifecycle, readiness tracking, and log output. This page covers all annotations available in Application templates.

Deployment stages

Nelm processes a deployment in three stages. Different annotation groups affect different stages.

1. Render

Templates are evaluated with the current values. No cluster access is needed (except for `lookup`). At this stage:

- `werf.io/deploy-on` controls whether a resource is included in the render output for the current operation type (`install`, `upgrade`, etc.).
- Template functions (`werf_secret_file`, `dump_debug`, etc.) are executed.
- `secret-values.yaml` and files from `secret/` are decrypted.

2. Plan

Nelm connects to the cluster, reads current resource state, and runs a **dry-run Server-Side Apply** to compute the exact diff. It then builds a **DAG of operations** — which resources to create, update, or delete, and in what order.

At this stage:

- Order and dependency annotations (`werf.io/weight`, `werf.io/deploy-dependency-*`, `*.external-dependency.werf.io/*`) shape the DAG.
- Lifecycle annotations (`werf.io/ownership`, `werf.io/delete-policy`, `werf.io/delete-propagation`) determine which operations enter the plan.

Dry-run SSA means the diff is computed by the API server — defaulting, admission plugins, and mutating webhooks are all applied.

3. Apply

The DAG is executed: resources are created, updated, or deleted in dependency order, with parallelism where no dependencies exist. Readiness tracking and log streaming run concurrently.

At this stage:

- Tracking annotations (`werf.io/track-termination-mode`, `werf.io/fail-mode`, `werf.io/failures-allowed-per-replica`, `werf.io/no-activity-timeout`) control waiting behavior.
- Log annotations control what is printed during deployment.

Common use cases

1. Job that must run before the main application

Classic case: database migration before the application starts.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: {{.Application.Instance.Name}}-db-migrate
  annotations:
    werf.io/delete-policy: before-creation
spec: ...
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{.Application.Instance.Name}}-app
  annotations:
    werf.io/deploy-dependency-migrate: state=ready,kind=Job,name=db-migrate
spec: ...
```

`before-creation` ensures the Job is recreated on each deploy (Kubernetes otherwise rejects updates to immutable Job fields). The Deployment waits for `ready` — i.e., successful job completion.

2. Preserve a resource across uninstall or chart removal

Use case: a PVC with database data that must not be deleted even when the Application is uninstalled.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: {{.Application.Instance.Name}}-postgres-data
  annotations:
    helm.sh/resource-policy: keep
spec:
  accessModes: [ReadWriteOnce]
  resources:
    requests:
      storage: 50Gi
  storageClassName: gp3
```

`helm.sh/resource-policy: keep` prevents the PVC from being deleted on uninstall or when removed from the chart. It does not protect against `d8 k delete pvc` — for that, use `persistentVolumeReclaimPolicy: Retain` on the StorageClass.

3. Resource shared between releases

A TLS Secret used by multiple charts should not disappear when any one of them is uninstalled.

```
apiVersion: v1
kind: Secret
metadata:
  name: {{.Application.Instance.Name}}-shared-tls
  annotations:
    werf.io/ownership: anyone
type: kubernetes.io/tls
data: ...
```

`anyone` tells Nelm it is not the sole owner, so it skips deletion on uninstall.

4. Wait for a resource created by an operator

Deploy only after cert-manager issues a certificate:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{.Application.Instance.Name}}-app
  annotations:
    cert.external-dependency.werf.io/resource: certificates.v1.cert-manager.io/myapp-tls
spec: ...
```

Nelm waits for the `Certificate` to be `present` and `ready` before creating the Deployment.

5. Non-critical component that must not fail the deploy

A metrics DaemonSet whose unavailability should not block the release:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: {{.Application.Instance.Name}}-metrics-agent
  annotations:
    werf.io/fail-mode: IgnoreAndContinueDeployProcess
    werf.io/track-termination-mode: NonBlocking
spec: ...
```

Nelm will not wait for readiness and will not fail on timeout.

6. Resource rendered only on first install

An init Job that runs only during `install`, not `upgrade`:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: {{.Application.Instance.Name}}-init-data
  annotations:
    werf.io/deploy-on: install
    werf.io/ownership: anyone   # CRITICAL
spec: ...
```

Without `werf.io/ownership: anyone`, on upgrade the resource renders as absent and the owning release removes it. `anyone` prevents that.

7. Slow-starting resource with a large image

StatefulSet with a large Docker image (ML models, Elasticsearch with pre-loaded indices):

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: elasticsearch
  annotations:
    werf.io/no-activity-timeout: 20m
    werf.io/failures-allowed-per-replica: "3"
    werf.io/show-service-messages: "true"
spec: ...
```

- `no-activity-timeout: 20m` — 20 minutes without events before Nelm considers it timed out. Use this for slow image pulls or long initialization.
- `failures-allowed-per-replica: "3"` — allows up to 3 restarts per replica before declaring failure. Use for flapping init dependencies.
- `show-service-messages: "true"` — emit Kubernetes Events in deploy output (useful for diagnosing `ImagePullBackOff`, scheduling failures, OOM events).

Ordering and dependencies

Weights

`werf.io/weight` groups resources: equal weights deploy in parallel; different weights deploy sequentially in ascending order.

```
metadata:
  annotations:
    werf.io/weight: "-10"   # Deploys before default-weight (0) resources
```

Direct dependencies

`werf.io/deploy-dependency-<id>` waits for a specific resource in the same release to reach a specific state before deploying the annotated resource.

```
metadata:
  annotations:
    werf.io/deploy-dependency-db: state=ready,kind=StatefulSet,name=postgres
    werf.io/deploy-dependency-migrations: state=present,kind=Job,name=db-migrate
```

Dependency states:

- `ready` — resource is in a ready state (e.g., Deployment's `availableReplicas == replicas`).
- `present` — resource exists in the cluster.

Full format:

```
werf.io/deploy-dependency-<id>: state=ready|present[,name=<name>][,namespace=<namespace>][,kind=<kind>][,group=<group>][,version=<version>]
```

⚠ This annotation has no effect if the dependency resource is in a different deploy stage (pre/main/post). Cross-stage ordering is already enforced by the stage sequence itself.

External dependencies

`<id>.external-dependency.werf.io/resource` waits for a resource **outside the release** (created by an operator or another release):

```
metadata:
  annotations:
    cert.external-dependency.werf.io/resource: certificates.v1.cert-manager.io/myapp-tls
    cert.external-dependency.werf.io/name: myapp-production  # Namespace of the external resource
```

Full format:

```
<id>.external-dependency.werf.io/resource: <kind>[.<version>.<group>]/<name>
```

Delete dependencies

`werf.io/delete-dependency-<id>` — the annotated resource will be deleted only after the referenced resource becomes `absent`:

```
metadata:
  annotations:
    werf.io/delete-dependency-app: state=absent,kind=Deployment,name=app
```

Lifecycle annotations

`helm.sh/resource-policy`

`keep` — do not delete the resource on uninstall or when removed from the chart. The resource continues to be updated on install/upgrade as long as it renders.

`werf.io/ownership`

- `release` (default for regular resources) — resource is deleted on uninstall and when absent from the chart. Release metadata annotations are applied.
- `anyone` (default for hooks and CRDs in `crds/`) — resource is not deleted on uninstall. Release metadata annotations are not applied.

Use `anyone` for resources shared between releases or for resources that should survive their release (init Jobs with `werf.io/deploy-on: install`).

`werf.io/deploy-on`

Controls on which lifecycle operations the resource is rendered.

```
werf.io/deploy-on: pre-install,upgrade,post-install
```

Allowed values: `pre-install`, `install`, `post-install`, `pre-upgrade`, `upgrade`, `post-upgrade`, `pre-rollback`, `rollback`, `post-rollback`, `pre-uninstall`, `uninstall`, `post-uninstall`. Default: `install,upgrade,rollback`.

> ⚠ If a resource is rendered for `install` but not `upgrade`, and `werf.io/ownership: release` is set, the resource will be **deleted on upgrade** because it is absent from the upgrade render. Set `werf.io/ownership: anyone` to prevent this.

`werf.io/delete-policy`

Controls when a resource is deleted relative to the apply operation.

Value	When
`before-creation`	Always recreate before apply
`before-creation-if-immutable`	Recreate only on “field is immutable” error (default for Jobs)
`succeeded`	Delete after successful deploy
`failed`	Delete on readiness-check failure

Values can be combined with a comma: `before-creation,succeeded`.

`werf.io/delete-propagation`

Kubernetes deletion propagation strategy.

Value	Behavior
`Foreground` (default)	Wait for dependent objects to be deleted
`Background`	Delete resource immediately; dependents deleted asynchronously
`Orphan`	Delete resource; leave dependents

Tracking annotations

Annotation	Default	Description
`werf.io/track-termination-mode`	`WaitUntilResourceReady`	`WaitUntilResourceReady` or `NonBlocking`
`werf.io/fail-mode`	`FailWholeDeployProcessImmediately`	`FailWholeDeployProcessImmediately` or `IgnoreAndContinueDeployProcess`
`werf.io/failures-allowed-per-replica`	`1`	Number of restarts per replica before declaring failure
`werf.io/no-activity-timeout`	`4m`	Go duration; timeout on absence of events or status changes
`werf.io/show-service-messages`	`false`	Show Kubernetes Events in deploy output

Log annotations

Annotation	Default	Description
`werf.io/skip-logs`	`false`	Suppress all pod logs
`werf.io/skip-logs-for-containers`	—	Comma-separated list of containers to suppress
`werf.io/show-logs-only-for-containers`	—	Comma-separated list; only show logs for these containers

Annotation	Default	Description
`werf.io/show-logs-only-for-number-of-replicas`	`1`	Show logs only for the first N replicas
`werf.io/log-regex`	—	RE2 pattern; show only matching lines
`werf.io/log-regex-skip`	—	RE2 pattern; suppress matching lines
`werf.io/log-regex-for-<container>`	—	Per-container RE2 filter for showing
`werf.io/log-regex-skip-for-<container>`	—	Per-container RE2 filter for suppressing

Template functions

`werf_secret_file`

Embed the decrypted content of a file from the `secret/` directory:

```
data:
  config.yaml: {{ werf_secret_file "config.yaml" | b64enc }}
```

Files in `secret/` are stored encrypted (AES-128-CBC, key from `NELM_SECRET_KEY`). They are decrypted in memory during render. Use for certificates, private keys, and large configs that don't fit in `secret-values.yaml`.

`dump_debug`, `printf_debug`, `include_debug`, `tpl_debug`

Debug-level output functions that emit to logs without affecting rendered output. Activated only when debug log level is set by the deploy system.

```
{{ dump_debug $ }}
{{ printf_debug "replicaCount: %d" .Values.replicaCount }}
{{ include_debug "myapp.labels" . | nindent 4 }}
{{ tpl_debug "{{ .Values.template }}" . }}
```

Known pitfalls

1. **`null` values and SSA.** Server-Side Apply frequently fails on fields with a `null` value. If `.Values.foo` is `nil`, the manifest gets `foo: null`. Fix with `{{ if .Values.foo }}` guards or `default`.
2. **`lookup` timing.** If cluster resources change between the Plan and Apply stages, the rendered plan may be stale. Avoid `lookup` for critical logic; pass data through values instead.
3. **Non-deterministic functions.** `now`, `randAlphaNum`, and map iteration with variable key ordering produce different renders on each call. With plan freezing this can create spurious diffs.
4. **`werf.io/deploy-dependency-*` and stages.** The annotation has no effect across deploy stages (pre/main/post). Cross-stage ordering is implicit in the stage sequence.

Hooks

Application hooks are written in Go using the same module-sdk as Deckhouse Kubernetes Platform (DKP) module hooks. The key difference is that Application hooks use `ApplicationHookInput` instead of `HookInput`, which adds Application-specific capabilities and enforces namespace isolation.

ApplicationHookInput

`ApplicationHookInput` exposes all the standard hook input methods plus:

- **`Instance()`** — returns metadata about the Application instance, including its name and namespace. Use this to scope operations to the correct namespace.
- **`ObjectPatcher()`** — returns a namespace-scoped patcher. Create and patch operations are limited to the namespace of the Application instance. Hooks cannot create or modify cluster-wide resources.

Example: basic hook

```
package main

import (
   "context"

   "github.com/deckhouse/module-sdk/pkg"
   applicationhook "github.com/deckhouse/module-sdk/pkg/app-hook"
)

func main() {
   applicationhook.Run(onSync)
}

func onSync(ctx context.Context, input applicationhook.ApplicationHookInput) error {
   instance := input.Instance()
   // instance.Name — name of the Application resource
   // instance.Namespace — namespace where the Application is installed

   patcher := input.ObjectPatcher()
   // patcher only operates within instance.Namespace

   return nil
}
```

Settings validation hook

Applications can include a settings validation hook that runs before DKP applies changes. This is used when OpenAPI schema validation is insufficient — for example, to check business logic constraints across multiple settings fields.

The hook implements a `Check` function with the signature:

```
func Check(_ context.Context, input settingscheck.Input) settingscheck.Result
```

`settingscheck.Input` provides access to the `Application.spec.settings` values.

`settingscheck.Result` is one of:

- `settingscheck.Allow(warnings...)` — settings are valid; optionally attach warning messages.
- `settingscheck.Reject(reason)` — settings are invalid; the Application resource will not be applied.

Example:

```
func Check(_ context.Context, input settingscheck.Input) settingscheck.Result {
   replicas := input.Settings.Get("replicas").Int()
   if replicas == 0 {
      return settingscheck.Reject("replicas cannot be 0")
   }

   var warnings []string
   if replicas == 2 {
      warnings = append(warnings, "an even number of replicas may cause split-brain in some
configurations")
   }

   if replicas > 3 {
      return settingscheck.Reject("replicas cannot be greater than 3")
   }

   return settingscheck.Allow(warnings...)
}
```

Namespace isolation guarantee

The `ObjectPatcher` returned by `ApplicationHookInput` enforces that:

- All `Create`, `Patch`, and `Update` calls target the Application’s own namespace.
- Calls targeting other namespaces or cluster-wide resources are rejected at runtime.

This is an architectural enforcement, not just a convention. Hooks that need to read (not write) cluster-wide resources can still use `d8 k` or the Kubernetes API client directly, but writes outside the namespace are

blocked.

Delivery subsystem

This section describes the architecture of the Delivery subsystem in Deckhouse Kubernetes Platform (DKP).

The Delivery subsystem includes the following modules:

- `operator-argo`: Manages Argo CD installations in DKP.
- `operator-helm`: Declaratively manages Helm chart deployments.

Operator-argo module

The `operator-argo` module deploys Argo CD Operator in a Deckhouse Kubernetes Platform (DKP) cluster. The module enables to install Argo CD in a DKP cluster using the ArgoCD resource.

The module works with the following custom resources:

- Application: Describes application deployment and its management.
- ApplicationSet: Provides templating and mass application creation according to defined rules.
- AppProject: Defines a set of applications and application access policies.
- ArgoCD: Main resource for deploying and configuring an Argo CD instance.
- ArgoCDExport: Exports Argo CD configuration and state for backup or migration.
- ImageUpdater: Automatically updates application container images.
- NamespaceManagement: Defines namespace management rules for an Argo CD instance.
- NotificationsConfiguration: Defines notification settings for events in Argo CD and applications.

For more details on module settings and usage examples, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.
> - Only the main containers of each component are shown in the diagram.

The Level 2 C4 architecture of the `operator-argo` module and its interactions with other DKP components are shown in the following diagrams:

- Main module operator:

- Argo CD instance deployment scenario with Redis in a non-HA configuration:

- Argo CD instance deployment scenario with Redis in an HA configuration (the diagram shows only differences from the primary deployment scenario):

- Argo CD instance deployment scenario in a principal cluster for a multicluster setup (the diagram shows only differences from the primary deployment scenario):

Module components

The module consists of the following components:

1. **Argocd-operator-controller-manager** (Deployment): Implementation of Argo CD Operator that allows deploying Argo CD instances in a DKP cluster. The component works with the following custom resources:
   - ArgoCD: Main resource for deploying and configuring an Argo CD instance.
   - ArgoCDExport: Exports Argo CD configuration and state for backup or migration. The operator reads the ArgoCDExport custom resource and creates a Job/CronJob with the same name as the ArgoCDExport resource. The created Job/CronJob performs backup of the Argo CD instance configuration.
   - NamespaceManagement: Defines namespace management rules for an Argo CD instance. The operator watches the NamespaceManagement custom resource and updates the `argocd-cmd-params-cm` ConfigMap accordingly.
   - NotificationsConfiguration: Defines notification settings for events in Argo CD and applications. The operator reads NotificationsConfiguration custom resources and updates configuration in the

`argocd-notifications-cm` ConfigMap based on them.

Argocd-operator-controller-manager creates Deployment, Secret, ConfigMap, StatefulSet, and other resources for each ArgoCD custom resource, adding that resource name as a prefix to the created resources.

It consists of the following containers:

- **manager**: Main container.
- **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to `manager` metrics.

> **i** The following components describe resources created by argocd-operator-controller-manager based on configuration defined in the ArgoCD custom resource. The `<ArgoCD name>` prefix is used in descriptions and is replaced by the controller with the ArgoCD resource name.

2. **<ArgoCD name>-server** (Deployment): Argocd-server. Main component for interacting with an Argo CD instance. Argocd-server provides REST/gRPC API and a web UI for Argo CD management. The component allows managing Application, ApplicationSet, and AppProject custom resources through the provided interfaces (web UI, API, CLI).

   The operator creates this component if the `.spec.server.enabled` parameter of the ArgoCD custom resource is set to `true` (default value is `true`).

   It consists of the following containers:

   - **argocd-server-init**: Optional set of init containers defined by the user in the `.spec.server.initContainers` parameter of the ArgoCD custom resource.
   - **rollout-extension**: Optional init container that loads the UI extension for the Rollout custom resource. The module does not provide a controller for this custom resource. Such controller must be installed and configured separately. Argocd-operator-controller-manager adds rollout-extension if `.spec.server.enableRolloutsUI` is set to `true`.
   - **argocd-server**: Main container.
   - **argocd-server-sidecar**: Optional set of sidecar containers defined by the user in the `.spec.server.sidecarContainers` parameter of the ArgoCD custom resource.

3. **<ArgoCD name>-repo-server** (Deployment): Argocd-repo-server. Component responsible for template rendering, application manifest generation, and working with external repositories used by Argo CD. Argocd-repo-server synchronizes application manifests from configured repositories and passes them to the corresponding components for further deployment.

   The operator creates this component if the `.spec.repo.enabled` parameter of the ArgoCD custom resource is set to `true` (default value is `true`).

   It consists of the following containers:

   - **copyutil**: Init container that copies executables for use by the main container.
   - **argocd-repo-server-init**: Optional set of init containers configured through the `.spec.repo.initContainers` parameter of the ArgoCD custom resource to prepare the environment.
   - **argocd-repo-server**: Main container that generates and processes manifests and works with remote application Git repositories.
   - **argocd-repo-server-sidecar**: Optional set of sidecar containers defined by the user in the `.spec.repo.sidecarContainers` parameter of the ArgoCD custom resource and used to extend repo-server functionality.

4. **<ArgoCD name>-application-controller** (StatefulSet): Argocd-application-controller. Component responsible for synchronization and state management of applications defined in Argo CD. Argocd-application-controller provides idempotent application of Kubernetes manifests, manages deployment, rollback, and self-healing workflows, and monitors resource state in the cluster.

   The operator creates this component if the `.spec.controller.enabled` parameter of the ArgoCD custom resource is set to `true` (default value is `true`).

   It consists of the following containers:

   - **application-controller-init**: Optional set of init containers configured through the `.spec.controller.initContainers` parameter of the ArgoCD custom resource to prepare the environment.
   - **argocd-application-controller**: Main container implementing synchronization logic for Application custom resources and resources created from them.
   - **application-controller-sidecar**: Optional set of sidecar containers defined by the user in the `.spec.controller.sidecarContainers` parameter of the ArgoCD custom resource and used to extend controller capabilities.

5. **<ArgoCD name>-applicationset-controller** (Deployment): Argocd-applicationset-controller. Optional component consisting of a single **applicationset-controller** container and responsible for managing the ApplicationSet custom resource in Argo CD. It allows automatic creation, update, and deletion of Application resources based on configured templates and generators (for example, Git, List, Matrix, and

Cluster generators). This simplifies mass management of similar applications that must be deployed across different environments or clusters.

The operator creates this component if the `.spec.applicationSet.enabled` parameter of the ArgoCD custom resource is set to `true` (default value is `true`).

For more details about the component, refer to the applicationset-controller documentation.

6. **<ArgoCD name>-argocd-image-updater-controller** (Deployment): Argocd-image-updater-controller. Optional component consisting of a single **argocd-image-updater** container and intended for automatic container image updates in Argo CD applications when new versions appear in image registries. The component tracks image tag changes and, when a new version is found, updates corresponding Application resources in Argo CD (for example, image tags in manifests or Helm values) via a pull request to a Git repository or directly, depending on the selected workflow.

   Argocd-image-updater-controller performs the following functions:

   - Manages the ImageUpdater custom resource that defines settings for automatic updates of application container images.
   - Periodically checks application container images in supported registries (Docker Hub, Quay.io, Harbor, and others).
   - Supports filtering image tags by patterns and update strategies (`semver`, `latest`, and others).
   - When a new image version is found, automatically performs write-back (writes the new image tag value) to Argo CD Application or a Git repository depending on the configured method.

   For correct operation, the component requires access to Git repositories and, if needed, private image registries. Credentials for accessing image registries can be stored in Kubernetes Secrets.

   To enable the component, set `.spec.imageUpdater.enabled` to `true` in the ArgoCD custom resource.

   For more details about the component, refer to the argocd-image-updater documentation.

7. **<ArgoCD name>-notifications-controller** (Deployment): Optional controller consisting of a single **argocd-notifications-controller** container that sends notifications about Argo CD events (for example, successful application sync, deployment failures, status changes, and others) to external notification systems, including email, Slack, Microsoft Teams, Telegram, OpsGenie, Webhook, and others.

   The main module operator argocd-operator-controller-manager generates notification settings based on NotificationsConfiguration custom resources and stores them in the `argocd-notifications-cm` ConfigMap and `argocd-notifications-secret` Secret used by the controller to generate and send notifications.

   To enable the component, set `.spec.notifications.enabled` to `true` in the ArgoCD custom resource.

   For more details on operation, refer to the Argo CD Notifications documentation.

8. **<ArgoCD name>-dex-server** (Deployment): Argocd-dex-server. Optional component for user authentication in Argo CD, acting as an OIDC provider (OpenID Connect) based on Dex. The component enables user login through various external authentication providers (LDAP, GitHub, GitLab, SAML, Azure AD, and others) and supports static users defined in Dex configuration.

   It consists of the following containers:

   - **copyutil**: Init container that copies executables for use by the main container.
   - **dex**: Main container.

   To enable the component, define parameters in `.spec.sso.dex` section of the ArgoCD custom resource.

   > ⚠ For Argo CD user authentication in DKP, the `operator-argo` module supports integration with the `user-authn` module (built-in DKP authentication). Other external providers via Dex are not used in this configuration.
   >
   > For more details about module usage examples, refer to the corresponding documentation section.

9. **<ArgoCD name>-redis** (Deployment): Argocd-regis. Mandatory component consisting of a single **redis** container and responsible for storing task queue data and session state in Argo CD. Argocd-regis provides a dedicated Redis database instance.

   Argocd-operator-controller-manager deploys this component if `.spec.ha.enabled` in the ArgoCD custom resource is `false`.

10. **<ArgoCD name>-redis-ha-server** (StatefulSet): Argocd-redis-ha-server. Mandatory component for deploying Redis in high availability (HA) mode in Argo CD. It provides a fault-tolerant Redis cluster with replication and automatic failover using Redis Sentinel.

    It consists of the following containers:

    - **config-init**: Init container that prepares configuration for Redis and Sentinel before main containers start.
    - **redis**: Main container implementing a Redis server instance.

- **sentinel**: Auxiliary container running Redis Sentinel to monitor Redis instance health and automatically switch to a replica if the primary instance fails.

Argocd-operator-controller-manager deploys this component if `.spec.ha.enabled` in the ArgoCD custom resource is `true`.

11. **<ArgoCD name>-redis-ha-haproxy** (Deployment): Argocd-redis-ha-proxy. Additional component for load balancing and distributing traffic to Redis cluster instances (redis-ha-server) based on the HAProxy load balancer/reverse proxy server.

    It consists of the following containers:

    - **config-init**: Init container that prepares HAProxy configuration before the main container starts.
    - **haproxy**: Container acting as a proxy server and providing transparent routing of client requests to available Redis master/replica instances, as well as automatic switching between them on failover.

    Argocd-operator-controller-manager deploys this component if `.spec.ha.enabled` in the ArgoCD custom resource is `true`.

    > **i**
    >
    > The following components are used to implement a secure and scalable multicluster setup in Argo CD:
    >
    > - Principal: The central control point. It stores state and distributes tasks.
    > - Agent: The executor in each target cluster. It applies manifests and reports back.
    >
    > This approach allows you to manage multiple clusters without direct access from the central Argo CD to the Kubernetes API of each cluster, which reduces the number of required open inbound connections, provides isolation, and ensures fault tolerance.

12. **<ArgoCD name>-agent-agent** (Deployment): Argocd-agent-agent. Optional component consisting of a single **<ArgoCD name>-agent-agent** container and responsible for executing operations on managed Kubernetes cluster resources based on requests from Argo CD. The component establishes connection to Argo CD Principal, synchronizes applications, and manages their state based on commands received from Argo CD Principal.

    For details on Argo CD multicluster architecture, refer to the Argo CD documentation.

    Argocd-operator-controller-manager deploys this component if `.spec.argoCDAgent.agent.enabled` in the ArgoCD custom resource is `true`. Argo CD Agent and Argo CD Principal cannot be enabled simultaneously in a single ArgoCD resource.

13. **<ArgoCD name>-agent-principal** (Deployment): Argocd-agent-principal. Optional component consisting of a single **<ArgoCD name>-agent-principal** container and enabling Argo CD operation in a multicluster setup.

    When this component is enabled, argocd-operator-controller-manager reconfigures all components that use Redis connection to use Redis proxy instead. The Argocd-agent-principal component provides this Redis proxy and routes database requests by analyzing Redis keys: depending on key values, a request is sent either to a local Redis instance or to one of remote Argo CD Agents.

    Argocd-operator-controller-manager deploys this component if `.spec.argoCDAgent.principal.enabled` in the ArgoCD custom resource is `true`. Argo CD Agent and Argo CD Principal cannot be enabled simultaneously in a single ArgoCD resource.

14. **<Export name>** (Job/CronJob): Argocd-export. Optional component implemented as Job or CronJob that creates a pod with a single **argocd-export** container. The component creates backup of Argo CD instance configuration and state.

Module interactions

The module interacts with the following components:

1. **External image registries**: Receives image lists.
2. **External code repositories**:
   - Receives application deployment manifests from repositories.
   - Updates `image` in Helm chart source code.
3. **External Argo CD Principal**:
   - Connects to Argo CD principal cluster.
   - Receives processing requests.
   - Sends back processing results.
4. **Kube-apiserver**:
   - Manages Application, ApplicationSet, AppProject, ArgoCD, ArgoCDExport, ImageUpdater, NamespaceManagement, NotificationsConfiguration custom resources, as well as Secret and ConfigMap.
   - Manages resources created during deployment of user applications described in the Application custom resource.
   - Authorizes requests for metrics retrieval.
5. **`user-authn` module**: Redirects user for authentication.

The following external components interact with the module:

1. **Prometheus-main**: Collects metrics provided by the operator and Argo CD instances.
2. **External Argo CD Agent**:
   - Connects to Argo CD principal cluster.
   - Receives processing requests.
   - Sends back processing results.

Additional resources

- Module `operator-argo` documentation
- Module `user-authn` documentation

Kubernetes & Scheduling subsystem

This subsection describes the architecture of the modules that are part of the Kubernetes & Scheduling subsystem of Deckhouse Kubernetes Platform (DKP).

The Kubernetes & Scheduling subsystem includes the following modules:

- `control-plane-manager`: Main module of the subsystem, responsible for managing cluster control plane components.
- `descheduler`: Analyzes the cluster state and evicts pods in accordance with the active strategies.
- `pod-reloader`: Allows you to automatically restart the workload if certain ConfigMap or Secret changes.
- `vertical-pod-autoscaler`: Automatically adjusts container resource requests and limits in pods based on actual consumption. The module architecture is described on the corresponding page.
- `adaptive-resource-management`: Automatically selects optimal resource requests and limits for containers using VPA (Vertical Pod Autoscaler) recommendations.

This subsection also describes the architecture of the control plane and the kubelet agent.

Cluster control plane

Deckhouse Kubernetes Platform (DKP) uses a standard (“vanilla”) Kubernetes cluster. The cluster control plane includes the following core components:

1. **kube-apiserver**: Kubernetes API server. It processes REST requests, provides access to the overall cluster state through which all other components interact, validates Kubernetes API resources, and stores them in **etcd** storage. It includes the following containers:
   - **kube-apiserver**: Main container.
   - **kube-apiserver-healthcheck**: Sidecar container that enables health checks of **kube-apiserver** without enabling anonymous authentication and without exposing an unauthenticated port. It uses a client certificate to authenticate to the API server. It is an open source project.
2. **etcd**: Distributed key-value storage that holds all configuration data and Kubernetes cluster resources.
3. **kube-scheduler**: Kubernetes scheduler. It analyzes node resources and assigns pods based on constraints and rules such as affinity and taints.
4. **kube-controller-manager**: Kubernetes controller manager. It runs controller loops that monitor and reconcile the state of standard Kubernetes resources to match the desired state. Examples of controllers shipped with Kubernetes: replication controller, endpoints controller, namespace controller, and ServiceAccount controller.

Interactions between the Kubernetes control plane components are shown in the architecture diagram of the `control-plane-manager` module.

Kubelet

Kubelet is not a control plane component, but it plays a key role in the operation of a Kubernetes cluster.

Kubelet is an agent that runs on every node in a Kubernetes cluster. It ensures that containers in pods are started and run according to their specifications. Kubelet continuously interacts with kube-apiserver to verify and maintain the state of nodes and containers. It is also responsible for starting control plane components.

Static pod manifests

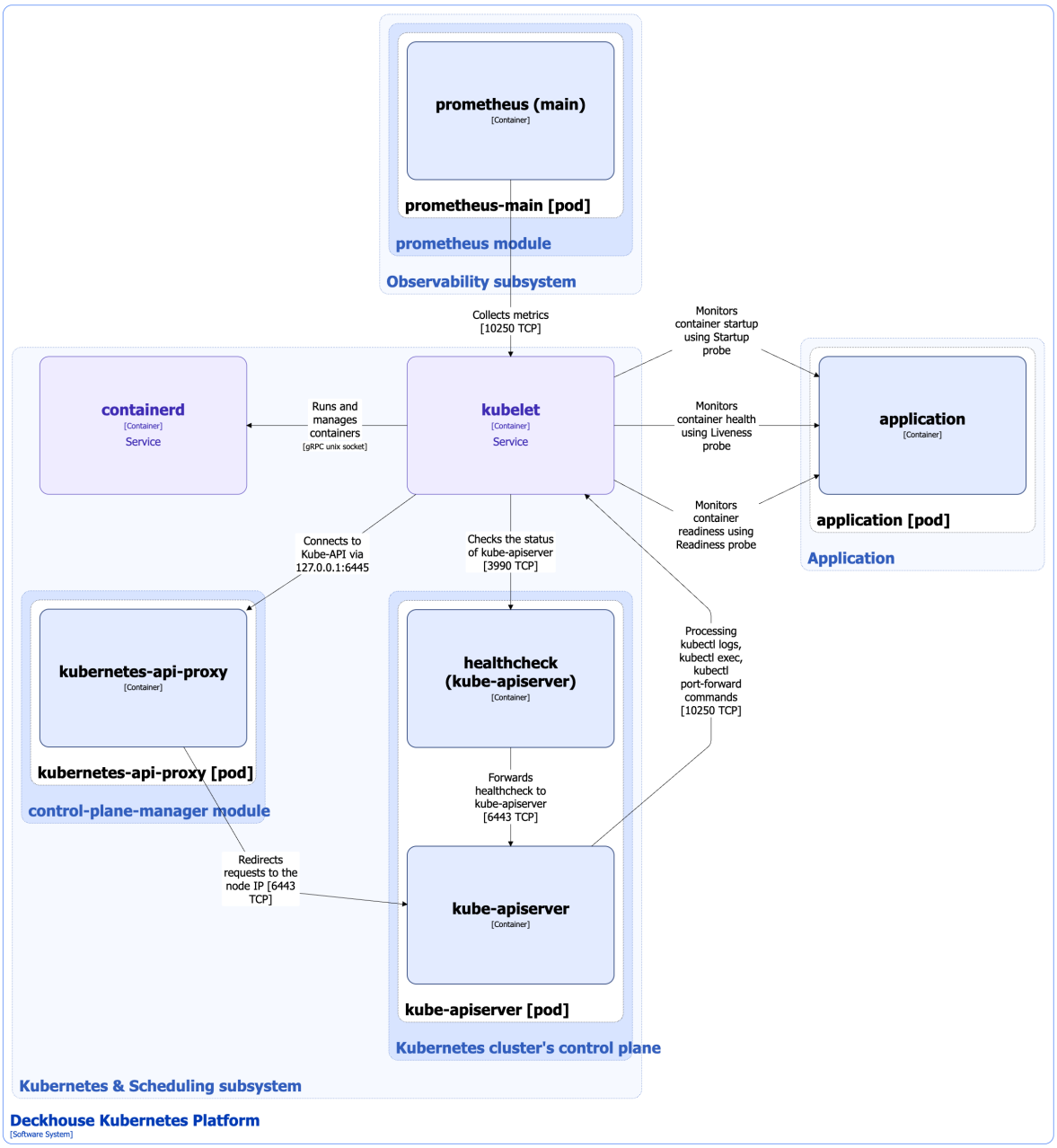
Kubelet starts control plane components from static pod manifests located in the `/etc/kubernetes/manifests` directory. In Deckhouse Kubernetes Platform, kubelet processes only files with the `.yaml` or `.yml` extension in this directory.

Files with other extensions, such as `kube-apiserver.backup`, `kube-apiserver.yaml.bak`, editor swap files, or other temporary files, are ignored. This prevents accidental processing of backup or non-manifest files as static pod manifests.

Kubelet interactions

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.
> - The diagram shows an `application` pod, which represents any pod in the cluster, both system and user-managed.

Kubelet interactions are shown in the following diagram:



Kubelet monitors the state of containers in all pods running on the node, including both user workloads and DKP components, by performing Startup, Liveness, and Readiness probes according to the pod specification. For more information about probes, see the Kubernetes documentation.

Kubelet also interacts with the following components:

1. **Containerd**: Receives commands from kubelet to manage the container lifecycle on the node via the Container Runtime Interface (CRI).
2. **Kubernetes-api-proxy**: Proxies requests to kube-apiserver that are sent to the `localhost` address. It is part of the `control-plane-manager` module.
3. **Kube-apiserver-healthcheck**: Checks the health of kube-apiserver.

The following components interact with kubelet:

1. **Kube-apiserver**:
   - Retrieving pod logs (the `kubectl logs` command).
   - Executing commands in running pods (the `kubectl exec` command).
   - Port forwarding (the `kubectl port-forward` command).
2. **Prometheus-main**: Collects kubelet metrics.

Additional resources

- Module `control-plane-manager` documentation

Cluster control plane management

Control-plane-manager module

Cluster control plane components are managed by the `control-plane-manager` module, which runs on all master nodes (nodes labeled with `node-role.kubernetes.io/control-plane: ""`).

Control plane management functions:

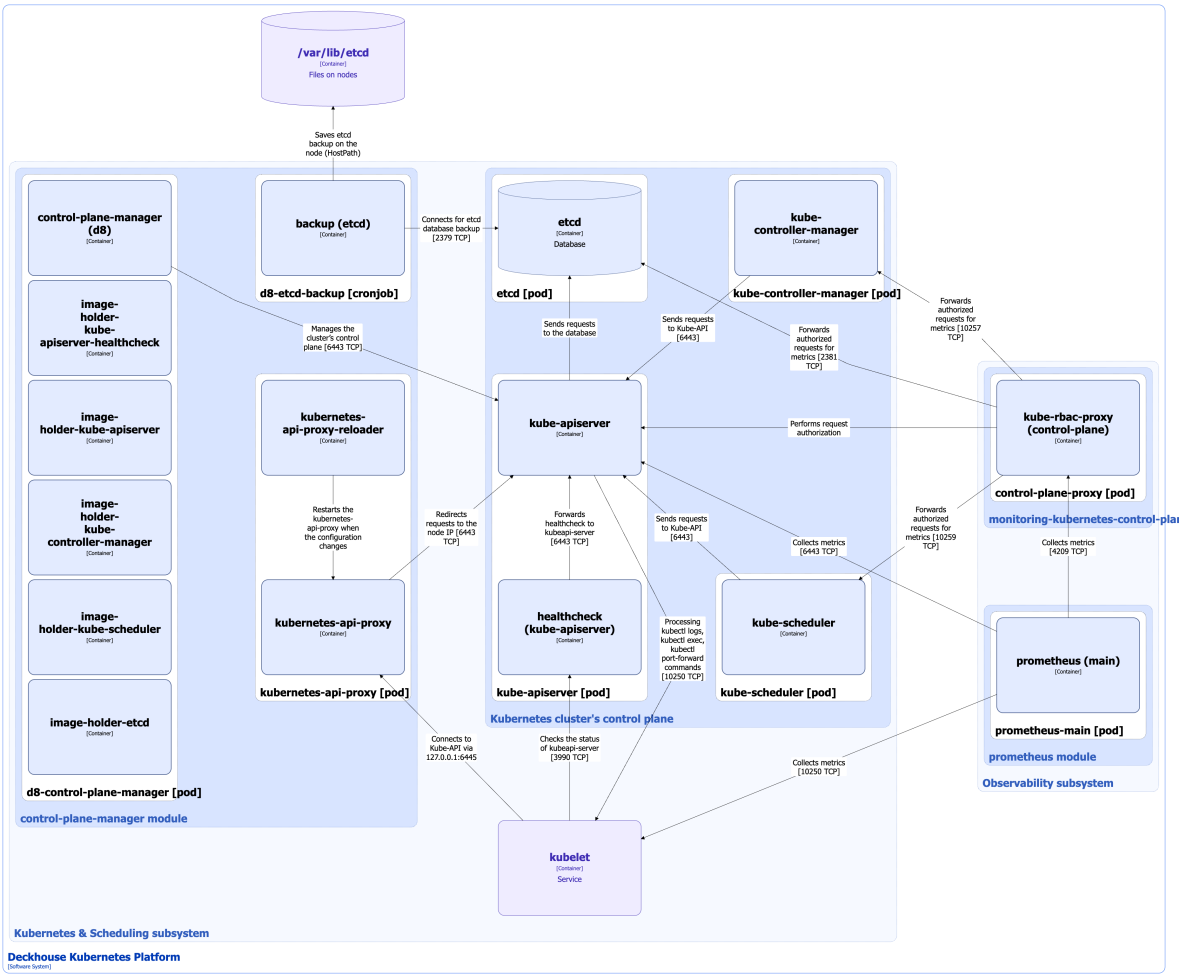
- **Certificate management**: Issuing, renewing, and rotating certificates required for control plane operation. Ensures automatic and secure control plane configuration and allows additional Subject Alternative Names (SAN) to be added for secure access to the Kubernetes API.
- **Component configuration**: Automatic generation of the required configuration files and manifests for control plane components.
- **Component upgrade and downgrade**: Maintains consistent component versions across the cluster.
- **Management of etcd cluster configuration**: Scaling master nodes and migrating between single-master and multi-master configurations.
- **Management of kubeconfig**: Maintains an up-to-date configuration for using `kubectl` on cluster nodes. Generates, renews, and updates the kubeconfig with *cluster-admin* privileges and creates a symbolic link for the `root` user so that the kubeconfig is used by default.
- **Scheduler extension**: Enables external plugins via webhooks using the KubeSchedulerWebhookConfiguration resource. This allows advanced scheduling logic when planning cluster loads, for example:
  - Placing data-intensive application pods closer to their data.
  - Prioritizing nodes based on their state (network load, storage subsystem health, etc.).
  - Dividing nodes into zones, etc.

For detailed configuration options and usage examples, refer to the `control-plane-manager` module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `control-plane-manager` module and its interactions with other platform components are shown in the following diagram:



Module components

The module consists of the following components:

1. **d8-control-plane-manager** (DaemonSet): Manages cluster control plane components and runs on all master nodes. It includes the following containers:
   - **control-plane-manager**: Main container. Developed by Flant.
   - A set of sidecar containers used to pre-pull images of control plane components. These containers remain paused and serve only as image holders:
     - **image-holder-kube-apiserver**
     - **image-holder-kube-apiserver-healthcheck**
     - **image-holder-kube-controller-manager**
     - **image-holder-kube-scheduler**
     - **image-holder-etcd**
2. **kubernetes-api-proxy** (static pods): Additional proxy server configured on each master node to handle requests to `localhost`. By default, it proxies requests to the local **kube-apiserver** instance. If the latter is unavailable, it sequentially queries the remaining **kube-apiserver** instances. It includes the following containers:
   - **kubernetes-api-proxy**: NGINX-based proxy server.
   - **kubernetes-api-proxy-reloader**: Sidecar container that restarts the proxy server when its configuration changes. Developed by Flant.
3. **d8-etcd-backup** (CronJob): Periodically performs backups of the cluster's **etcd** database. It consists of the following container:
   - **backup**: Container running a shell script that creates an etcd snapshot using `etcdctl` and stores it in `/var/lib/etcd` on the master node (default directory, configurable via the module parameters).

Module interactions

The module interacts with the following components:

1. **kube-apiserver**:
   - Manages cluster control plane components.
   - Proxies and load-balances requests to **kube-apiserver** sent to `localhost`.

2. **etcd**:
    - Manages etcd cluster configuration and membership.
    - Performs periodic database backups.

The following external components interact with the module:

- **kubelet**: Requests to **kube-apiserver** sent to `localhost` are proxied by the module's **kubernetes-api-proxy** component.

Cluster control plane monitoring

Control plane monitoring is provided by the `monitoring-kubernetes-control-plane` module, which ensures secure metrics collection and provides a basic set of monitoring rules for the following cluster components:

- **kube-apiserver**
- **kube-controller-manager**
- **kube-scheduler**
- **etcd**

For configuration details, refer to the `monitoring-kubernetes-control-plane` module documentation.

Components of the monitoring-kubernetes-control-plane module

The module consists of a single component:

1. **control-plane-proxy** (DaemonSet): Runs on all master nodes and includes a single container:
    - **kube-rbac-proxy**: Authorization proxy based on Kubernetes RBAC that provides secure access to metrics.

Interactions of control-plane-proxy component

Control-plane-proxy interacts with the following components:

1. **kube-apiserver**: Authorizes requests for metrics.
2. Control plane components: **control-plane-proxy** forwards authorized metric requests to:
    - **kube-controller-manager**
    - **kube-scheduler**
    - **etcd**

Prometheus-main interacts with **control-plane-proxy** to collect control plane component metrics.

The interaction between the `monitoring-kubernetes-control-plane` module and the cluster control plane is shown in the architecture diagram of the `control-plane-manager` module above.

Metrics collection from kube-apiserver

Metrics from **kube-apiserver** are collected directly by **prometheus-main**. The `monitoring-kubernetes-control-plane` module adds the corresponding metric collection rules to the **prometheus-main** configuration.

Additional resources

- Module `control-plane-manager` documentation
- Module `monitoring-kubernetes-control-plane` documentation

Vertical Pod Autoscaler

The `vertical-pod-autoscaler` module provides Vertical Pod Autoscaler (VPA) in Deckhouse Kubernetes Platform (DKP).

For details on module configuration and usage examples, refer to the relevant documentation section.

VPA operating modes

VPA can operate in two modes:

- Automatic resource adjustment:
    - **InPlaceOrRecreate** (the default in Kubernetes starting from version 1.33): VPA attempts to update resources without recreating Pods. If in-place resource updates are not possible, VPA falls back to behavior similar to the **Recreate** mode: the Pod for which the resources cannot be updated is evicted, and the controller creates a new Pod with updated resources.

> **i** To use the **InPlaceOrRecreate** mode in Kubernetes versions earlier than 1.33, enable the `InPlacePodVerticalScaling` feature gate in the `control-plane-manager` configuration.

  - **Auto** (the default in Kubernetes versions earlier than 1.33): VPA changes resource requests without recreating Pods but behaves the same as **Recreate** and restarts the Pod when necessary. This is a deprecated operating mode that will no longer be supported in future Deckhouse Kubernetes Platform (DKP) versions.
  - **Recreate**: VPA adjusts the resources of running Pods by restarting them. For a single Pod (`replicas: 1`), this will result in service unavailability during the restart. VPA does not restart Pods that were created without a controller.

- Recommendations only, without modifying resources:
  - **Initial**: Pod resources are adjusted only when Pods are created and not during the runtime.
  - **Off**: VPA does not change resources automatically. However, it still provides resource recommendations, which can be viewed using the `d8 k describe vpa` command.

When VPA is enabled and configured, resource requests are set automatically based on Prometheus data. You can also configure the system to only provide recommendations without applying any changes. For details on enabling and configuring the VPA, refer to Administration.

VPA limitations

Before using the vertical pod autoscaler (VPA), you need to consider several limitations:

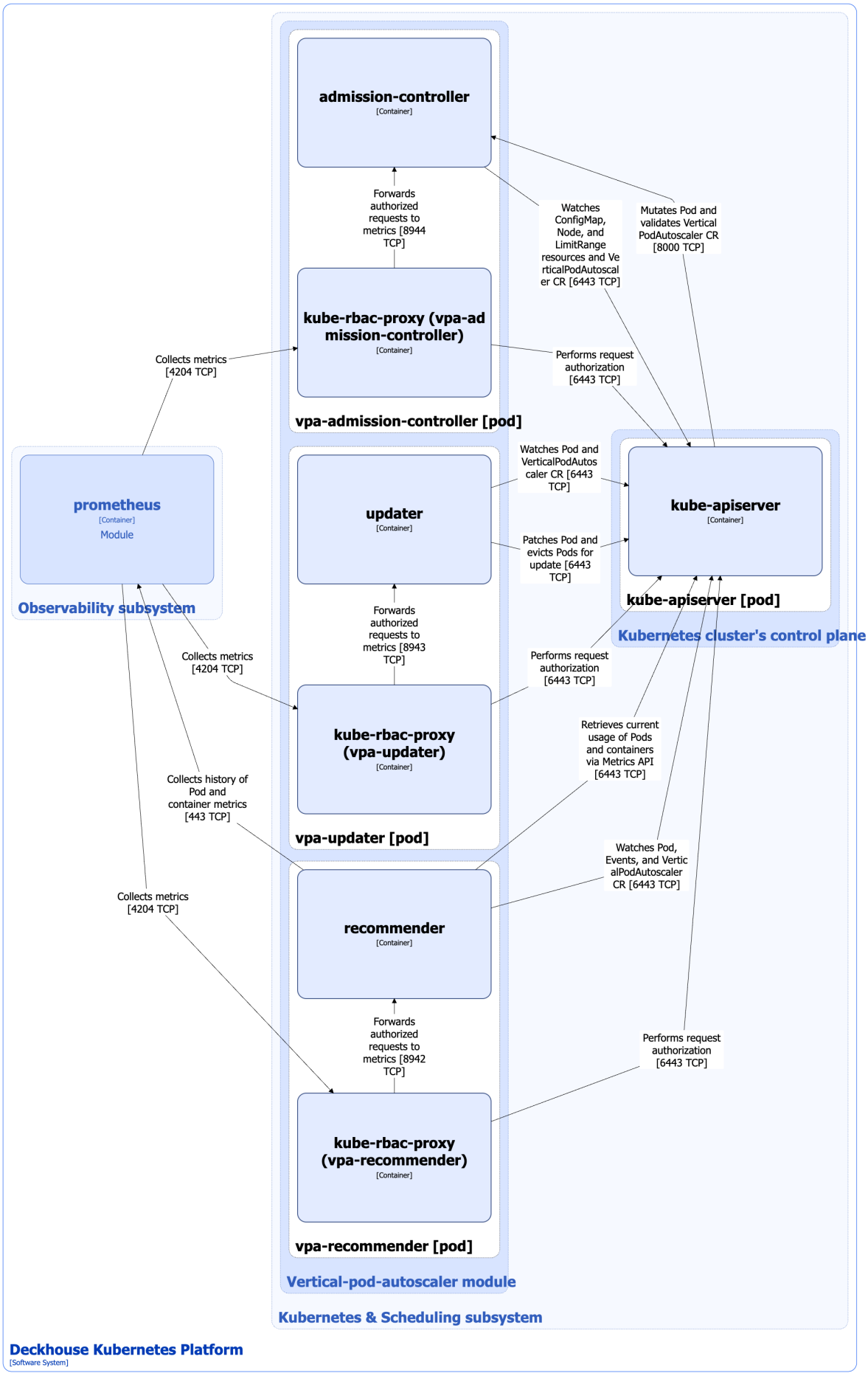
- Pod restarts when resources change:
  - Updating requested resources is an experimental feature. Each time VPA changes the resources, it recreates the pod, which may then be scheduled on a different node.
  - Pods can be rescheduled on other nodes.
- Compatibility with Horizontal Pod Autoscaler (HPA):
  - It is not recommended to use VPA together with HPA that is set for scaling based on CPU and memory.
  - VPA can be used with HPA that is set for scaling based on custom or external metrics.
- Issues in large clusters: VPA can work in large clusters, but the load on VPA increases as the number of pods grows.
- Issues with Pending pods: VPA may recommend resources that exceed those available in the cluster, causing pods to get stuck in the `Pending` state.
- Problems when deleting VPA: If you delete or disable VPA (Off mode), the resources will remain at the last modified values. This can cause confusion when resource values differ between the Helm charts, the controller, and the pods themselves.
- Using multiple VPA resources for a single pod: This can lead to unpredictable behavior.

> **⚠** When using VPA, it is recommended to configure a Pod Disruption Budget.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `vertical-pod-autoscaler` module and its interactions with other DKP components are shown in the following diagram:



Module components

The `vertical-pod-autoscaler` module consists of the following components:

1. **Vpa-admission-controller** (Deployment): A VPA controller that handles the VerticalPodAutoscaler

custom resource.

The vpa-admission-controller component performs the following actions:

- Validates VerticalPodAutoscaler custom resources.
- When a Pod is created and the VPA mode is not Off, the controller automatically sets or updates `requests` and `limits` in containers, optimizing them according to the current recommendations. The controller updates `limits` values only if the `spec.resourcePolicy.containerPolicies.controlledValues` parameter in the resource management policy is set to `RequestsAndLimits`.

It consists of the following containers:

- **admission-controller**: Main container.
- **kube-rbac-proxy**: Sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to admission-controller metrics. It is an open source project.

2. **Vpa-updater** (Deployment): A VPA component that checks whether pods with VPA have the correct resource settings. Vpa-updater performs in-place resource updates through the `pods/resize` Kubernetes subresource; if that is not possible or does not fit the resource management policy, it evicts the Pod.

   It consists of the following containers:

   - **updater**: Main container.
   - **kube-rbac-proxy**: Sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to updater metrics. It is an open source project.

3. **Vpa-recommender** (Deployment): A VPA component that calculates recommendations for `requests` based on past and current pod resource consumption.

   Vpa-admission-controller and vpa-updater recalculate `limits` values proportionally to `requests` values if the `spec.resourcePolicy.containerPolicies.controlledValues` parameter in resource management policy is set to `RequestsAndLimits`.

   It consists of the following containers:

   - **recommender**: Main container.
   - **kube-rbac-proxy**: Sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to recommender metrics. It is an open source project.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches standard resources such as ConfigMap, Node, LimitRange, and Pod, as well as VerticalPodAutoscaler and VerticalPodAutoscalerCheckpoint custom resources.
   - Retrieves current resource consumption through the Metrics API.
   - Evicts running pods when their resource specifications do not match the recommended values.
   - Authorizes requests for metrics.
2. **Prometheus**: Retrieves the history of pod resource consumption metrics.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Validates VerticalPodAutoscaler custom resources.
   - Changes `requests` and `limits` in the Pod specification.
2. **Prometheus**: Collects module metrics.

Additional resources

- Enabling vertical scaling
- Module `control-plane-manager` parameters
- Module `vertical-pod-autoscaler` documentation

Adaptive-resource-management module

The `adaptive-resource-management` module lets you automate the selection of resource requests and limits for workloads by leveraging the Vertical Pod Autoscaler (VPA) recommendations.

The module deploys the **AutoVPA** controller based on Goldilocks tailored for Deckhouse Kubernetes Platform (DKP). The controller automatically creates and maintains VPA objects for workloads in selected namespaces and provides recommendations on resource configuration.

Main features:

- Automatic creation of VPA objects for Deployments, StatefulSets, DaemonSets, and Jobs in managed

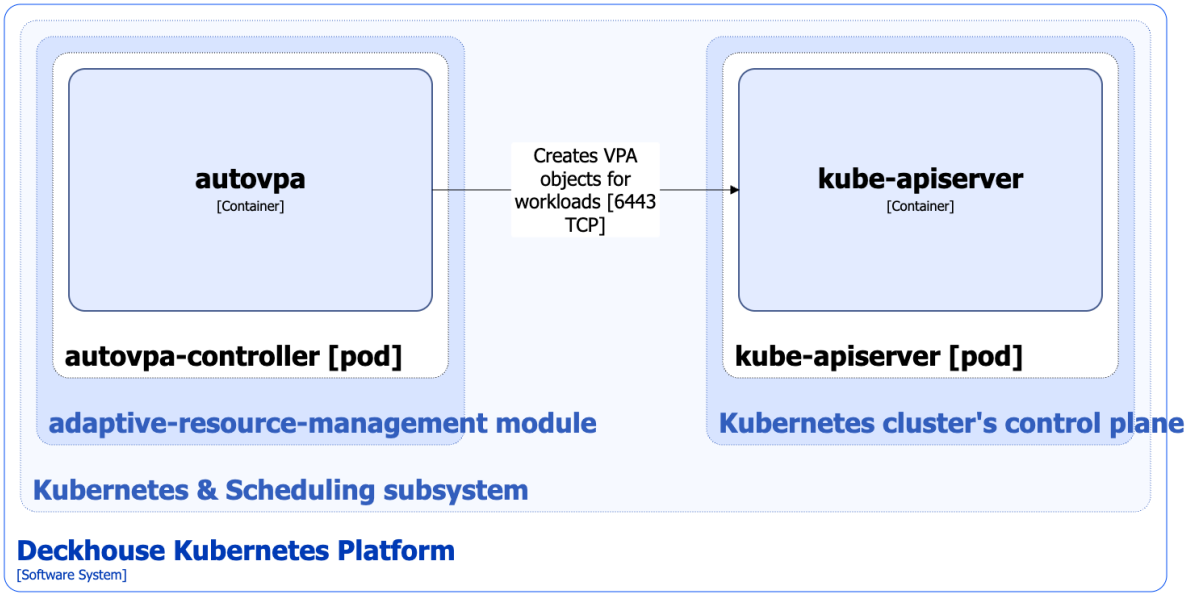
namespaces.
- Flexible namespace selection: manage all namespaces, only namespaces with a specific label, or namespaces matching a label selector.
- Combined namespace selection mode using a label selector together with the special label `autovpa.deckhouse.io/enabled`.
- Automatic creation of VPA objects in recommendation-only mode without changing the workload manifests.
- Per-workload and per-namespace VPA tuning via `autovpa.deckhouse.io/*` annotations and labels.
- Minimal resource footprint: the controller runs as a single replica and requires low CPU and memory resources.

For more details about the module configuration and usage examples, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `adaptive-resource-management` module and its interactions with other components of DKP are shown in the following diagram:



Module components

The `adaptive-resource-management` module consists of a single **autovpa-controller** component that includes **autovpa**, the main container.

Module interactions

The module interacts with the **kube-apiserver** component for creating VPA objects for workloads.

Additional resources

- Module `adaptive-resource-management` documentation

Descheduler module

The `descheduler` module ensures operation of Descheduler in Deckhouse Kubernetes Platform (DKP).

The module analyzes the cluster state periodically and evicts pods that match conditions defined in active strategies. Evicted pods are then scheduled again according to the current cluster state. This helps redistribute workloads in line with the selected strategy.

If the Metrics API service is registered in the cluster, the module uses it to obtain information about the current resource consumption.

Strategy parameters are configured using the Descheduler custom resource. A module hook processes these resources and recreates configuration for Descheduler.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `descheduler` module and its interactions with other DKP components are shown in the following diagram:

Module components

The `descheduler` module consists of a single **descheduler** component that includes the following containers:

- **descheduler**: Main container.
- **kube-rbac-proxy**: Sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to descheduler metrics.

Module interactions

The module interacts with the **kube-apiserver** component:

- Watches standard Node and Pod resources.
- Retrieves current resource consumption through the Metrics API.
- Evicts running pods to match active strategies.
- Authorizes requests for metrics.

The **prometheus** component interacts with the module by collecting module metrics.

Additional resources

- Module `descheduler` documentation

Pod-reloader module

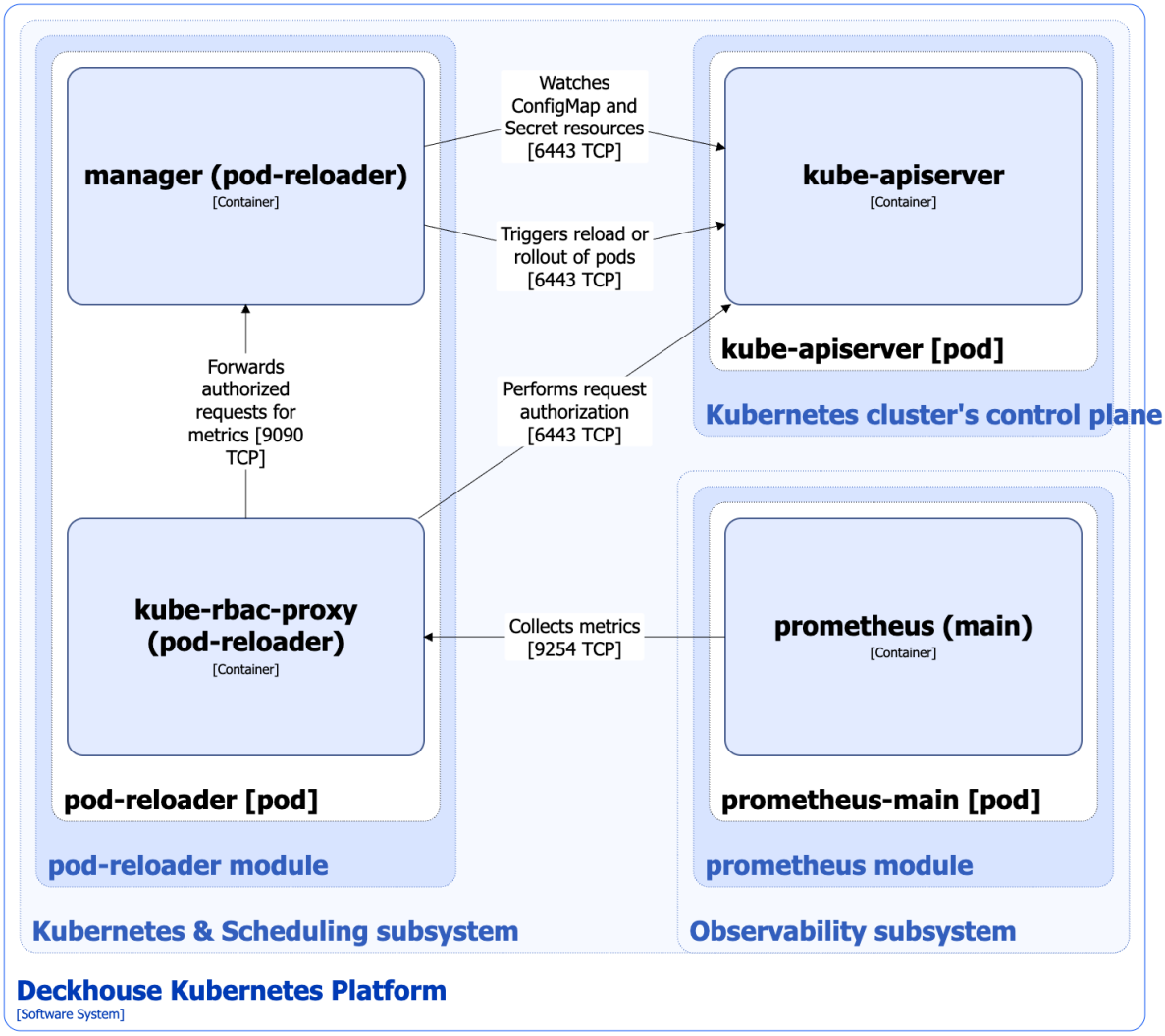
The `pod-reloader` module is based on Reloader. It provides the ability for automatic rollout on ConfigMap or Secret changes. Annotations are used for configuration. The module is running on **system** nodes.

For more details about module configuration and usage examples, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `pod-reloader` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The `pod-reloader` module consists of a single `pod-reloader` component that includes the following containers:

- **manager**: Main container.
- **kube-rbac-proxy**: Sidecar container with a Kubernetes RBAC-based authorization proxy that provides secure access to pod-reloader metrics. It is an open-source project.

Module interactions

The module interacts with the `kube-apiserver` component:

- Watches ConfigMap and Secret resources.
- Triggers reload or rollout of pods.
- Authorizes requests for metrics.

The `prometheus-main` component interacts with the module by collecting module metrics.

Additional resources

- Module `pod-reloader` documentation

Cluster & Infrastructure subsystem

This section describes the architecture of the Cluster & Infrastructure subsystem of the Deckhouse Kubernetes Platform (DKP).

The Cluster & Infrastructure subsystem is responsible for the infrastructure layer of Kubernetes cluster management. Cluster node management is implemented using the `node-manager` module, while interaction with IaaS providers is handled by the corresponding `cloud-provider-` family modules.

This section describes:

- The mechanisms for managing all node types used in DKP, as well as hybrid node groups and clusters.
- Bashible service, which is a key component of the Cluster & Infrastructure subsystem. Bashible is used by the `node-manager` module to manage node configuration.

The Cluster & Infrastructure subsystem also includes the following modules:

- `chrony`: Provides time synchronization across all cluster nodes.
- `registry-packages-proxy`: Provides an internal proxy server for container registry packages.
- `terraform-manager`: Provides tools for managing Terraform state within a Kubernetes cluster.

- Modules for cloud providers supported by DKP:
  - `cloud-provider-aws`
  - `cloud-provider-azure`
  - `cloud-provider-dvp`
  - `cloud-provider-dynamix`
  - `cloud-provider-gcp`
  - `cloud-provider-huaweicloud`
  - `cloud-provider-openstack`
  - `cloud-provider-vcd`
  - `cloud-provider-vsphere`
  - `cloud-provider-yandex`
  - `cloud-provider-zvirt`

Node management

Node types and addition mechanics

In Deckhouse Kubernetes Platform (DKP), nodes are divided into the following types:

- **CloudEphemeral**: Nodes are automatically provisioned, created, and deleted in the configured cloud provider.
- **CloudPermanent**: Persistent nodes created and updated by the `node-manager` module. The nodes differ in that their configuration comes not from the NodeGroup custom resource, but from a special resource <PROVIDER>ClusterConfiguration (for example, AWSClusterConfiguration for AWS). Also, an important difference is that to apply a node configuration, you need to run `dhctl converge` (by running the DKP installer). An example of a CloudPermanent node in a cloud cluster is a cluster master node.
- **CloudStatic**: Created manually or by any external tools, located in the same cloud that is integrated with one of the cloud provider modules:
  - CloudStatic nodes have several features related to integration with the cloud provider. These nodes are managed by the `cloud-controller-manager` component, resulting in:
    - Zone and region metadata being automatically added to the Node object.
    - When the virtual machine is deleted in the cloud, the corresponding Node object is also removed from the cluster.
    - CSI driver can be used to attach cloud volumes.
- **Static**: Static node hosted on a bare-metal server or a virtual machine. In the case of a cloud environment, the `cloud-controller-manager` component does not manage such a node — even if one of the cloud providers is enabled. To learn more about working with static nodes, refer to the `node-manager` documentation.

Nodes are added to the cluster by creating a NodeGroup object, which describes the type, parameters, and configuration of the node group. For CloudEphemeral groups, DKP interprets this object and automatically creates the corresponding nodes, registering them in the Kubernetes cluster. For other types (e.g., CloudPermanent or Static), node creation and registration must be done manually or via external tools.

Hybrid groups are also supported, where a single NodeGroup can include both Static nodes deployed in the cloud and static (bare-metal or virtual machine) nodes. For example, the main load may be handled by bare-metal servers, while cloud instances are used as scalable additions during peak loads.

Automatic deployment, configuration, and update of Kubernetes nodes

Automatic deployment (in *static/hybrid* — partial), configuration, and software updates work in all clusters regardless of being cloud or bare metal.

Kubernetes node deployment

DKP automatically deploys cluster nodes by performing the following **idempotent** operations:

- OS setup and optimization for working with `containerd` and Kubernetes:
  - Required packages are installed from distribution repositories.
  - Kernel parameters, logging settings, log rotation, and other system parameters are configured.
- Installation of required versions of `containerd` and kubelet, and registration of the node in the Kubernetes cluster.
- Kubernetes-api-proxy setup and updating the upstream list for balancing requests from the node to the Kubernetes API.

Maintaining node state

Two types of updates can be applied to maintain the node’s up-to-date state:

- **Regular**: Always applied automatically and do not cause downtime or reboot.
- **Disruptive**: Such as kernel or `containerd` updates, significant kubelet version changes, etc. These can be configured in manual or automatic mode via the `disruptions` section. In automatic mode, a node drain is performed before the update.

Only one node per group is updated at a time, and only if all nodes in the group are available.

DKP includes built-in monitoring metrics to track update progress, notify of issues, or prompt for manual approval if needed.

Working with nodes in supported clouds

Each supported cloud provider allows automatic node provisioning. You need to specify required parameters for each node or group.

Depending on the provider, parameters may include:

- Node type or CPU/memory capacity
- Disk size
- Security settings
- Connected networks, etc.

VM creation, startup, and cluster joining are performed automatically.

Cloud node scaling

Two node scaling modes are available:

- **Automatic scaling.**

  When resources are insufficient and pods are in `Pending` state, nodes are added to the group. If there is no load on one or several nodes, they are removed from the cluster. With automatic scaling, a group priority is considered (groups with higher priority are scaled first).

  To enable auto-scaling, specify different non-zero values for `minPerZone` and `maxPerZone` number of nodes per group.

- **Fixed node count.**

  DKP will maintain the specified number of nodes (for example, by provisioning new nodes to replace failed ones).

  To disable auto-scaling and maintain a fixed count, use the same value for `minPerZone` and `maxPerZone`.

Working with Static nodes

When working with static nodes, node-manager functions are limited as follows:

- **No node provisioning.** Resource allocation (bare-metal servers, VMs, etc.) is manual. Further configuration (joining the cluster, monitoring, etc.) can be fully or partially automated.
- **No auto-scaling.** Maintaining node count in a group is available using Cluster API Provider Static (via the `staticInstances.count` parameter). DKP tries to keep the specified node count in the group, removing excessive nodes and configuring the new ones (from StaticInstance resources in *Pending* state).

Node configuration, draining, adding to the cluster and removal can be performed in one of the following ways:

- **Manually** using prepared scripts.

  To configure a server (VM) and add a node to the cluster, download and run a bootstrap script. It’s generated for each static NodeGroup and stored in the secret `d8-cloud-instance-manager/manual-bootstrap-for-<NODEGROUP-NAME>`.

  To remove a node from the cluster and wipe the server (VM), use the `/var/lib/bashible/cleanup_static_node.sh` script (already present on each static node).

- **Automatically** using Cluster API Provider Static.

  CAPS connects to the server (VM) using StaticInstance and SSHCredentials resources, configures and adds the node to the cluster.

  If needed (e.g., if a corresponding StaticInstance resource is removed or the node count is decreased), CAPS connects to the cluster node, drains it, and removes from the cluster.

- **Manually, with transition to automatic management** via CAPS.

> **i** Available since DKP 1.63.

To transfer an existing cluster node under CAPS control, prepare StaticInstance and SSHCredentials as for automatic management described above, and annotate the StaticInstance resource with `static.node.deckhouse.io/skip-bootstrap-phase: ""`.

Node grouping and group management

Grouping and managing nodes as a group means that all group nodes will share metadata from the `NodeGroup` custom resource.

Monitoring features for node groups include:

- Grouped node parameter charts
- Grouped node availability alerts
- Alerts for N unavailable nodes or N% group nodes, etc.

What is the Instance resource

The Instance resource in Kubernetes represents a definition of an ephemeral virtual machine object, without a specific implementation. It's an abstraction used to manage machines created by tools such as `MachineControllerManager` or Cluster API Provider Static.

The object has no specification. Its status includes:

1. Reference to InstanceClass if available.
2. Reference to the Kubernetes Node object.
3. Current machine status.
4. Info on how to check machine creation logs (available during machine creation).

Creating or deleting a machine will create or delete the corresponding Instance resource. You cannot manually create an Instance resource, but you can delete it. This will remove the machine from the cluster (implementation-dependent).

When node reboot is required

Some node configuration changes may require a reboot.

For example, a node reboot is needed after modifying `sysctl` parameters like `kernel.yama.ptrace_scope`.

NodeGroup parameter effects on node updates and restarts

NodeGroup parameter	Disruption update	Node reprovisioning	Kubelet restart
chaos	-	-	-
cloudInstances.classReference	-	+	-
cloudInstances.maxSurgePerZone	-	-	-
cri.containerd.maxConcurrentDownloads	-	-	+
cri.type	- (NotManaged) / + (other)	-	-
disruptions	-	-	-
kubelet.maxPods	-	-	+
kubelet.rootDir	-	-	+
kubernetesVersion	-	-	+
nodeTemplate	-	-	-
static	-	-	+
update.maxConcurrent	-	-	-

For full details, see the NodeGroup custom resource documentation.

If InstanceClass or `instancePrefix` values change in the DKP configuration, no `RollingUpdate` will occur. Instead, new MachineDeployment objects will be created and old ones removed. The number of simultaneously provisioned MachineDeployments is defined by the `cloudInstances.maxSurgePerZone` parameter.

When an update requires a node disruption, a pod eviction process is initiated. If a pod cannot be evicted, the

eviction attempt is retried every 20 seconds for up to 5 minutes global timeout. After that, pods that have not been evicted are forcefully removed.

Additional resources

- Module `cloud-provider-aws` provider configuration
- Module `node-manager` documentation

Bashible

Bashible scripts and the bashible service

Node management functions are implemented using specially prepared bash scripts called **bashible**. The service that runs on cluster nodes and executes these scripts is also called bashible. A collection of scripts is referred to as a bundle.

4 bundles are used:

- Bashible installation scripts
- Bootstrap scripts for the first node
- Node configuration scripts for a specific cloud provider (for example, AWS)
- Common scripts

The scripts are implemented as *gotemplate* templates, which allows flexible node configuration depending on the node group. The scripts must be written in such a manner so that they can be safely re-executed in case of failure or repeated runs. An individual script is referred to as a step.

Main node configuration stages:

- Configuring the NodeUser to ensure access to the node.
- Installing CA certificates.
- Creating the `/opt/deckhouse/bin` directory, adding it to `PATH`, and placing required binaries there.
- Downloading required packages from `registrypackages`.
- Installing and configuring the containerd CRI.
- Downloading and configuring **kubernetes-api-proxy**. This component provides access to the Kubernetes API and is implemented as an NGINX instance with upstream servers pointing to master nodes. This ensures high availability of the API in case one master node is unavailable, as well as load balancing.
- Installing, configuring, and starting kubelet.
- Starting the bashible service, which runs `bashible.sh` every minute.
- Rebooting the node if required.

Bashible-api-server

Due to the large number of bashible script variations for different supported operating systems, storing all versions in etcd is not feasible because of key size limitations and the additional load on etcd. For this reason, the **bashible-api-server** component was developed to generate bashible scripts from templates stored in Custom Resources.

Bashible-api-server is a Kubernetes Extension API Server deployed on master nodes.

When kube-apiserver receives a request for resources containing bashible bundles, it forwards the request to bashible-api-server and returns the generated result. The interactions between bashible and bashible-api-server are shown in the architecture diagrams of the `node-manager` module (for example, in the CloudEphemeral nodes diagram).

Bashible-api-server returns the following resources:

- The **second-phase bootstrap script** downloaded during the first phase.
- **Bashibles**: The `bashible.sh` script.
- **Nodegroupbundles**: Rendered bundle containing the set of scripts for bootstrapping and configuring a node.

These resources can be retrieved either via the Kubernetes API or using `kubectl` by specifying the node group name:

- `kubectl get bootstrap.bashible.deckhouse.io master -o yaml`
- `kubectl get bashibles.bashible.deckhouse.io master -o yaml`
- `kubectl get nodegroupbundles.bashible.deckhouse.io master -o yaml`

Bashible-api-server calculates a checksum of all scripts associated with a node group. This is required to implement the update mechanism and ensure correct node group status updates. The checksums are stored in the `d8-cloud-instance-manager/configuration-checksums` secret. A checksum change triggers the re-execution of bashible scripts on nodes when the configuration is modified. In addition, the checksum of the bashible service is reset every 4 hours to force periodic re-execution of bashible.

Node-manager module

Cluster nodes are managed by the `node-manager` module.

For a detailed description of the module’s functions, configuration options, and usage examples, refer to the corresponding documentation section.

Module architecture

The module architecture varies depending on the node type and differs in terms of its component composition. The following pages describe the module architecture for different node types:

- Managing CloudEphemeral nodes
- Managing CloudPermanent nodes
- Managing CloudStatic nodes
- Managing Static nodes
- Managing hybrid node groups and clusters

Additional resources

- Module `node-manager` documentation

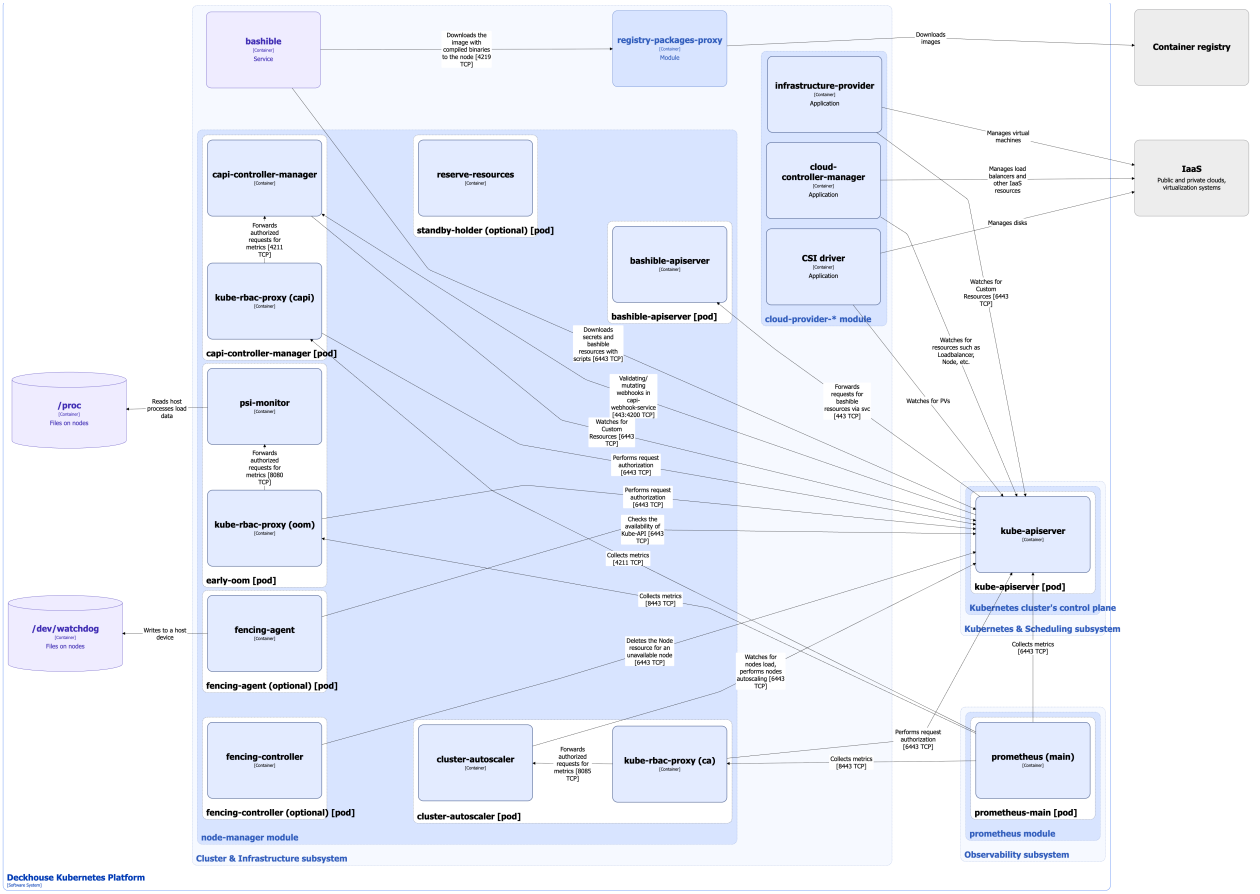
CloudEphemeral node management

This page describes the architecture of the `node-manager` module for CloudEphemeral nodes.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `node-manager` module and its interactions with other Deckhouse Kubernetes Platform (DKP) components are shown in the following diagram:



Module components

> **i** Bashible is a key component of the Cluster & Infrastructure subsystem that enables the operation of the `node-manager` module. However, it is not part of the module itself, as it runs at the OS level as a system service. For Bashible details, refer to the corresponding documentation section.

The module managing CloudEphemeral nodes consists of the following components:

1. **Bashible-api-server**: A Kubernetes Extension API Server deployed on master nodes. It generates bashible scripts from templates stored in custom resources. When kube-apiserver receives a request for resources containing bashible bundles, it forwards the request to bashible-api-server and returns the generated result. For more details about bashible and bashible-api-server, refer to the corresponding documentation section.
2. **Capi-controller-manager** (Deployment): Core controllers from the Kubernetes Cluster API project. Cluster API extends Kubernetes to manage clusters as custom resources within another Kubernetes cluster. The capi-controller-manager pod consists of the following containers:
   - **control-plane-manager**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to controller metrics.
3. **Cluster-autoscaler** (Deployment): An additional Kubernetes component that automatically adjusts the number of nodes in the cluster based on workload. For more details, refer to the node management documentation section.

   The component includes:
   - **cluster-autoscaler**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cluster-autoscaler metrics.
4. **Early-oom** (DaemonSet): A pod deployed on every node. It reads resource load metrics from `/proc` and terminates pods under high load before kubelet does. Enabled by default, but can be disabled in the module configuration if it causes issues for normal node operation.

   Includes the following containers:
   - **psi-monitor**: Monitors the *PSI (Pressure Stall Information)* metric, which reflects how long processes wait for resources such as CPU, memory, or I/O.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the early-oom metrics.
5. **Fencing-agent** (DaemonSet) and **fencing-controller**: Components that implement the fencing mechanism. The operation principles of both components are described in detail in the `spec.fencing.mode` parameter description of the NodeGroup resource. For details on how the fencing mechanism handles different node types, refer to FAQ in the `node-manager` module documentation.
6. **Standby-holder** (Deployment): A pod used to reserve nodes. When the `spec.cloudinstances.standby` parameter is enabled in the NodeGroup custom resource, standby nodes are created in all configured zones.

   A standby node is a cluster node with pre-reserved resources available for immediate scaling. This allows cluster-autoscaler to schedule workloads without waiting for node initialization, which may take several minutes.

   The standby-holder pod does not perform useful work. It simply reserves resources to prevent cluster-autoscaler from deleting temporarily unused nodes.

   The pod has the lowest PriorityClass and is evicted when real workloads are scheduled. For details on pod priority and preemption, refer to the Kubernetes documentation.

   The pod includes a single container **reserve-resources**.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Retrieves the `kube-system/d8-node-manager-cloud-provider` Secret for cloud connectivity.
   - Works with Cluster API custom resources.
   - Manages Node resources.
   - Monitors node load.
   - Performs node autoscaling.
   - Authorizes metric requests.
2. Node filesystem:
   - `/proc`: Reads PSI metrics for OOM handling.
   - `/dev/watchdog`: Sends signals to reset the Watchdog timer.

> **i** The module interacts with the `cloud-provider` module via kube-apiserver using the `kube-system/d8-node-manager-cloud-provider` Secret to obtain cloud connection settings and create CloudEphemeral nodes. The `cloud-provider` module also provides provider-specific Cluster API custom resource templates to `node-manager`.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Executes mutating and validating webhooks of capi-controller-manager.
   - Forwards requests for bashible resources to bashible-api-server.
2. **Prometheus-main**:
   - Collects metrics from `node-manager` module components.

Architecture features specific to CloudEphemeral nodes

1. Nodes are ephemeral and automatically created and deleted by the module.
2. A configured cloud provider module (`cloud-provider-*`) is required for interaction with cloud infrastructure. It also includes csi-driver and cloud-controller-manager.
3. **Capi-controller-manager** manages the lifecycle of the cluster and its nodes through higher-level custom resources, without directly provisioning infrastructure. It generates infrastructure-specific custom resources, leaving provisioning to the `cloud-provider` module.
4. **Cluster-autoscaler** enables node autoscaling.
5. Node reservation is supported.

Additional resources

- Module `node-manager` documentation

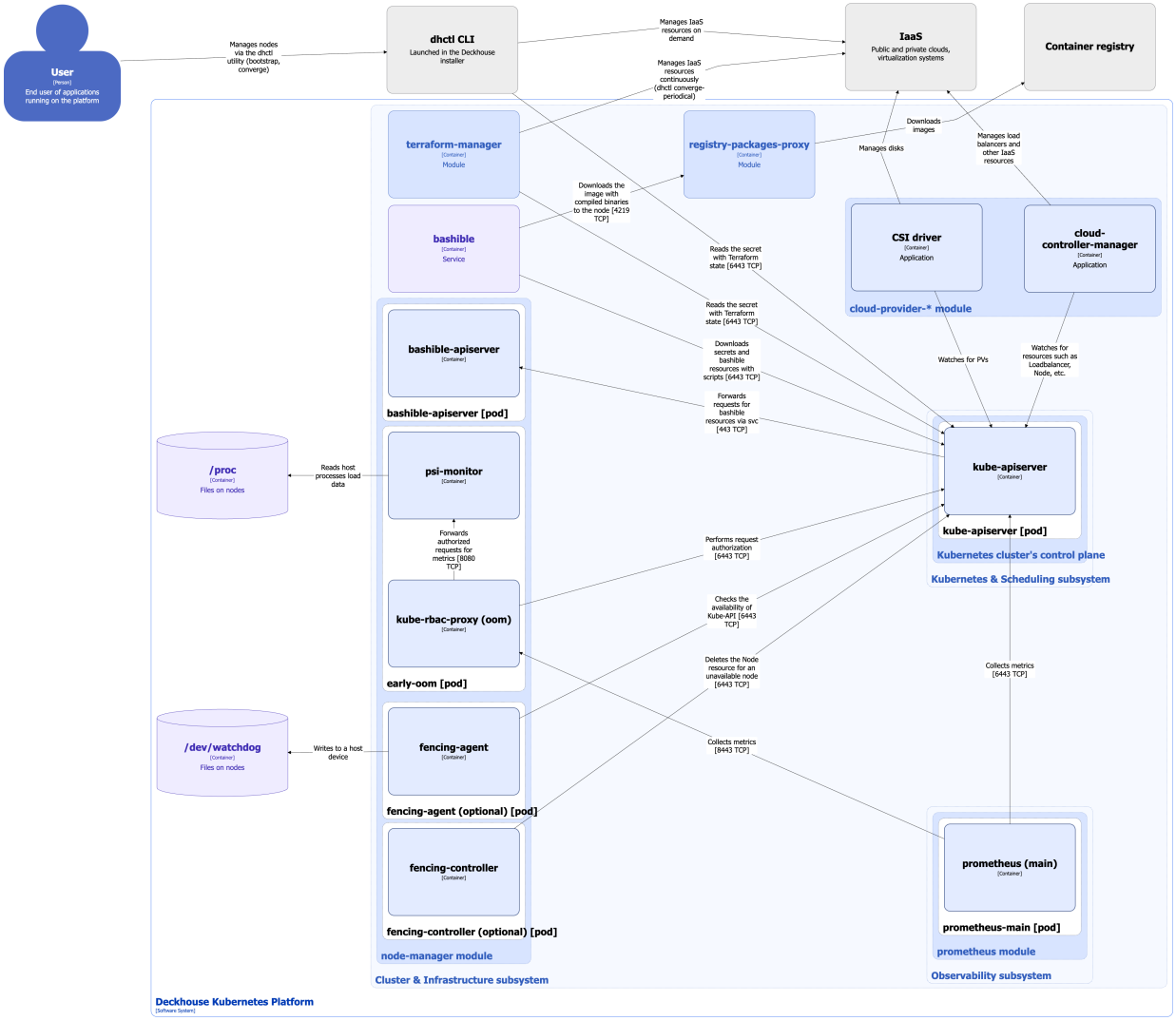
CloudPermanent node management

This page describes the architecture of the `node-manager` module for CloudPermanent nodes.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `node-manager` module and its interactions with other Deckhouse Kubernetes Platform (DKP) components are shown in the following diagram:



Module components

> **i** Bashible is a key component of the Cluster & Infrastructure subsystem that enables the operation of the `node-manager` module. However, it is not part of the module itself, as it runs at the OS level as a system service. For Bashible details, refer to the corresponding documentation section.

The module managing CloudPermanent nodes consists of the following components:

1. **Bashible-api-server**: A Kubernetes Extension API Server deployed on master nodes. It generates bashible scripts from templates stored in custom resources. When kube-apiserver receives a request for resources containing bashible bundles, it forwards the request to bashible-api-server and returns the generated result. For more details about bashible and bashible-api-server, refer to the corresponding documentation section.
2. **Early-oom** (DaemonSet): A pod deployed on every node. It reads resource load metrics from `/proc` and terminates pods under high load before kubelet does. Enabled by default, but can be disabled in the module configuration if it causes issues for normal node operation.

   Includes the following containers:

   - **psi-monitor**: Monitors the *PSI (Pressure Stall Information)* metric, which reflects how long processes wait for resources such as CPU, memory, or I/O.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the early-oom metrics.
3. **Fencing-agent** (DaemonSet) and **fencing-controller**: Components that implement the fencing mechanism. The operation principles of both components are described in detail in the `spec.fencing.mode` parameter description of the NodeGroup resource. For details on how the fencing mechanism handles different node types, refer to FAQ in the `node-manager` module documentation.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

    - Manages Node resources.
    - Authorizes metric requests.

2. Node filesystem:

    - `/proc`: Reads PSI metrics for OOM handling.
    - `/dev/watchdog`: Sends signals to reset the Watchdog timer.

The following external components interact with the module:

1. **Kube-apiserver**:

    - Forwards requests for bashible resources to bashible-api-server.

2. **Prometheus-main**:

    - Collects metrics from `node-manager` module components.

Architecture features specific to CloudPermanent nodes

1. Nodes are persistent and are created, managed, and deleted by the user. Node management is performed not directly in the infrastructure but via the **dhctl** utility executed as part of the DKP installer.
2. `Terraform-manager` is a module used for automated management of cloud infrastructure resources. It checks the Terraform state and applies non-destructive changes to infrastructure resources. The module architecture is described on the corresponding documentation page.
3. **Csi-driver** is used to provision disks in the cloud infrastructure.
4. **Cloud-controller-manager** is used to provision load balancers and other infrastructure resources according to its specification.
5. **Infrastructure-provider** is not required. All node management operations are performed by the user via the **dhctl** utility and the `terraform-manager` module.
6. Automatic node scaling is not supported.

Additional resources

- Module `node-manager` documentation
- Module `terraform-manager` documentation

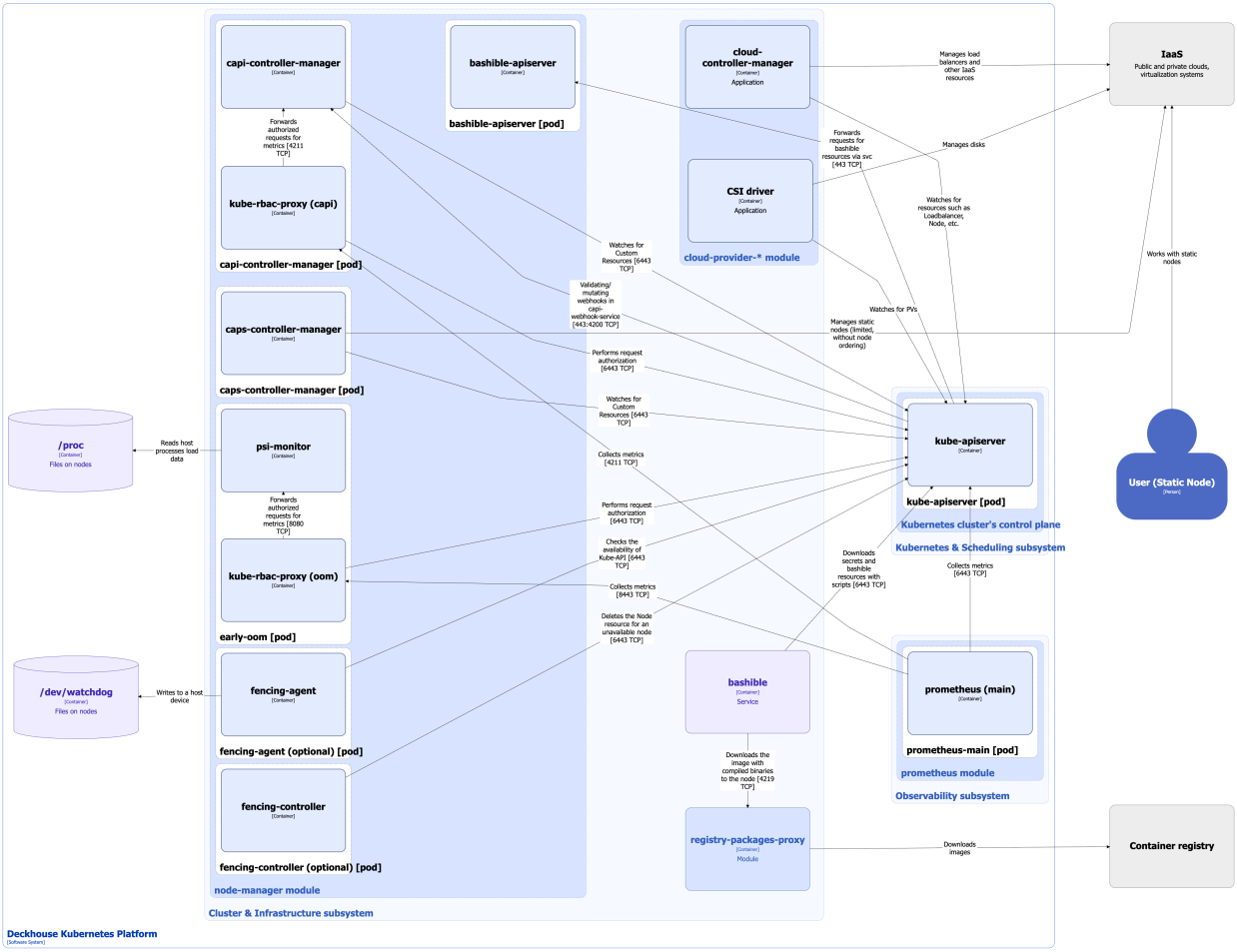
CloudStatic node management

This page describes the architecture of the `node-manager` module for CloudStatic nodes.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `node-manager` module and its interactions with other Deckhouse Kubernetes Platform (DKP) components are shown in the following diagram:



Module components

> **i** Bashible is a key component of the Cluster & Infrastructure subsystem that enables the operation of the `node-manager` module. However, it is not part of the module itself, as it runs at the OS level as a system service. For Bashible details, refer to the corresponding documentation section.

The module managing CloudStatic nodes consists of the following components:

1. **Bashible-api-server**: A Kubernetes Extension API Server deployed on master nodes. It generates bashible scripts from templates stored in custom resources. When kube-apiserver receives a request for resources containing bashible bundles, it forwards the request to bashible-api-server and returns the generated result. For more details about bashible and bashible-api-server, refer to the corresponding documentation section.
2. **Capi-controller-manager** (Deployment): Core controllers from the Kubernetes Cluster API project. Cluster API extends Kubernetes to manage clusters as custom resources within another Kubernetes cluster. The capi-controller-manager pod consists of the following containers:
   - **control-plane-manager**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to controller metrics.
3. **Caps-controller-manager** (Deployment): CAPI Provider Static (CAPS), an implementation of a provider for declarative management of static nodes (bare-metal servers or virtual machines) in the Kubernetes Cluster API project. It operates as an extension to capi-controller-manager.

   CAPS provides an additional abstraction layer over the existing DKP mechanism for automatic configuration and cleanup of static nodes using scripts generated for each node group. The component is not tied to a specific cloud provider. For more details, refer to the `node-manager` documentation.
4. **Early-oom** (DaemonSet): A pod deployed on every node. It reads resource load metrics from `/proc` and terminates pods under high load before kubelet does. Enabled by default, but can be disabled in the module configuration if it causes issues for normal node operation.

   Includes the following containers:
   - **psi-monitor**: Monitors the *PSI (Pressure Stall Information)* metric, which reflects how long processes wait for resources such as CPU, memory, or I/O.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access

to the early-oom metrics.

5. **Fencing-agent** (DaemonSet) and **fencing-controller**: Components that implement the fencing mechanism. The operation principles of both components are described in detail in the `spec.fencing.mode` parameter description of the NodeGroup resource. For details on how the fencing mechanism handles different node types, refer to FAQ in the `node-manager` module documentation.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Works with Cluster API custom resources.
   - Manages Node resources.
   - Authorizes metric requests.
2. Node filesystem:
   - `/proc`: Reads PSI metrics for OOM handling.
   - `/dev/watchdog`: Sends signals to reset the Watchdog timer.
3. Infrastructure:
   - Manages static nodes (partially, without provisioning).

The following external components interact with the module:

1. **Kube-apiserver**:
   - Executes mutating and validating webhooks of capi-controller-manager.
   - Forwards requests for bashible resources to bashible-api-server.
2. **Prometheus-main**:
   - Collects metrics from `node-manager` module components.

Architecture features specific to CloudStatic nodes

1. Users create and configure nodes in the following ways:
   - Manually, using bashible scripts preconfigured in DKP.
   - Manually, with the following node handover to CAPS for automated management.
   - Automatically, using CAPS.
2. **Capi-controller-manager** manages the lifecycle of the cluster and its nodes through higher-level custom resources, without directly provisioning infrastructure. It generates infrastructure-specific custom resources, leaving provisioning to the infrastructure provider (CAPS).
3. **Caps-controller-manager**: Component managing static nodes (partially, without provisioning).
4. **Csi-driver** is used to provision disks in the cloud infrastructure.
5. **Cloud-controller-manager** is used to provision load balancers and other infrastructure resources according to its specification.
6. **Infrastructure-provider** of a specific cloud is not required, as CAPS acts on its behalf.
7. Automatic node scaling is not supported.

Additional resources

- Module `node-manager` documentation

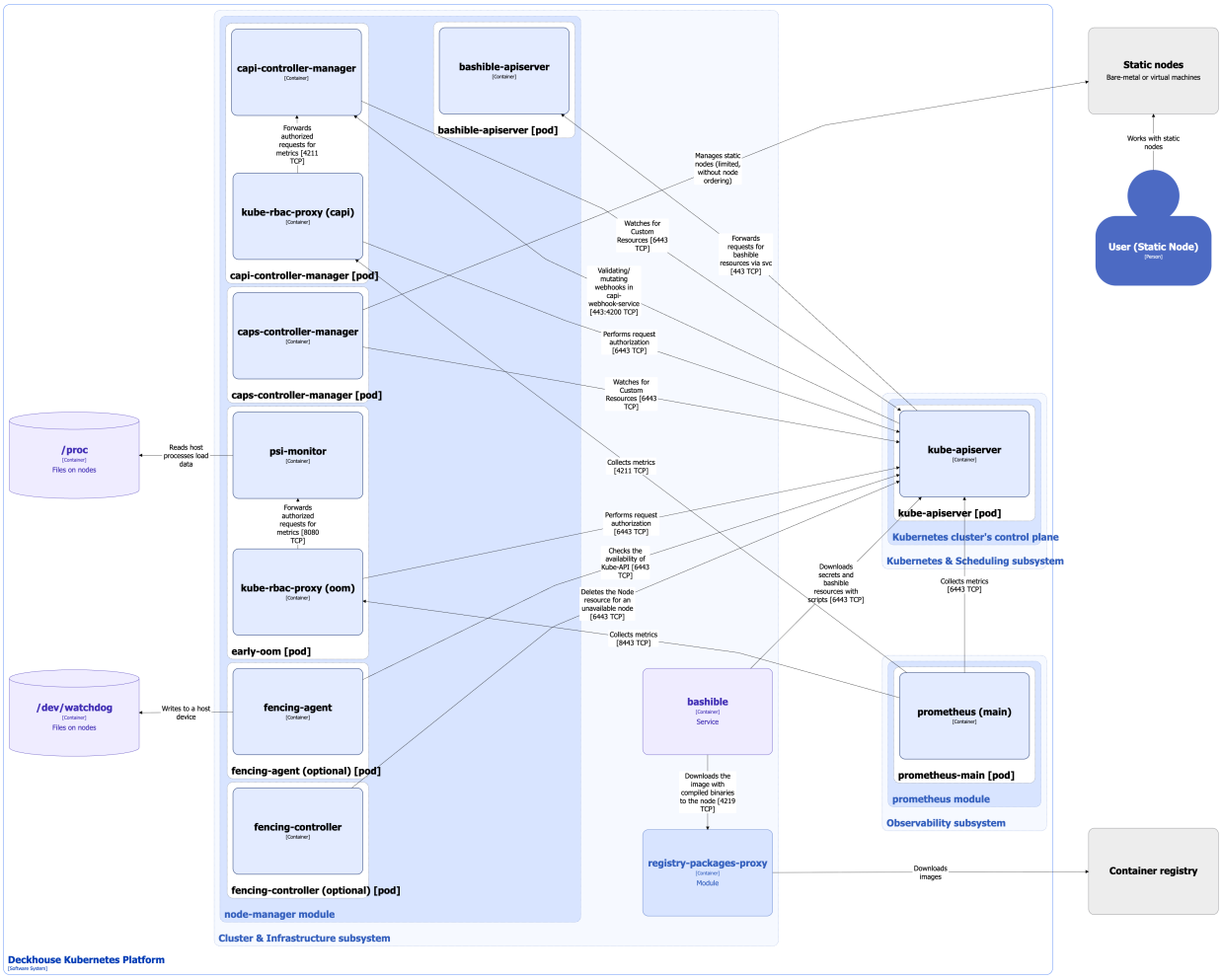
Static node management

This page describes the architecture of the `node-manager` module for Static nodes.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `node-manager` module and its interactions with other Deckhouse Kubernetes Platform (DKP) components are shown in the following diagram:



Module components

> **i** Bashible is a key component of the Cluster & Infrastructure subsystem that enables the operation of the `node-manager` module. However, it is not part of the module itself, as it runs at the OS level as a system service. For Bashible details, refer to the corresponding documentation section.

The module managing Static nodes consists of the following components:

1. **Bashible-api-server**: A Kubernetes Extension API Server deployed on master nodes. It generates bashible scripts from templates stored in custom resources. When kube-apiserver receives a request for resources containing bashible bundles, it forwards the request to bashible-api-server and returns the generated result. For more details about bashible and bashible-api-server, refer to the corresponding documentation section.

2. **Capi-controller-manager** (Deployment): Core controllers from the Kubernetes Cluster API project. Cluster API extends Kubernetes to manage clusters as custom resources within another Kubernetes cluster. The capi-controller-manager pod consists of the following containers:

   - **control-plane-manager**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to controller metrics.

3. **Caps-controller-manager** (Deployment): CAPI Provider Static (CAPS), an implementation of a provider for declarative management of static nodes (bare-metal servers or virtual machines) in the Kubernetes Cluster API project. It operates as an extension to capi-controller-manager.

   CAPS provides an additional abstraction layer over the existing DKP mechanism for automatic configuration and cleanup of static nodes using scripts generated for each node group. The component is not tied to a specific cloud provider. For more details, refer to the `node-manager` documentation.

4. **Early-oom** (DaemonSet): A pod deployed on every node. It reads resource load metrics from `/proc` and terminates pods under high load before kubelet does. Enabled by default, but can be disabled in the module configuration if it causes issues for normal node operation.

   Includes the following containers:

   - **psi-monitor**: Monitors the *PSI (Pressure Stall Information)* metric, which reflects how long processes wait for resources such as CPU, memory, or I/O.

    - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the early-oom metrics.

5. **Fencing-agent** (DaemonSet): Deployed to a node group when the `spec.fencing` parameter of the NodeGroup custom resource is enabled.

   The operation principles of the component is described in detail in the `spec.fencing.mode` parameter description of the NodeGroup resource. For details on how the fencing mechanism handles different node types, refer to FAQ in the `node-manager` module documentation.

   Consists of a single container:

    - **fencing-agent**: Performs the necessary checks and writes to `/dev/watchdog` to signal the watchdog.

6. **Fencing-controller**: A controller that watches all nodes labeled with `node-manager.deckhouse.io/fencing-enabled`.

   If a node is unavailable for more than 60 seconds, the controller deletes all pods from the node but **does not delete the Node object** for static nodes (`node.deckhouse.io/type=Static`). This preserves the node's cluster registration so it can return to service after being manually restored.

   For details on how the fencing mechanism handles different node types, refer to FAQ in the `node-manager` module documentation.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
    - Works with Cluster API custom resources.
    - Manages Node resources.
    - Authorizes metric requests.
2. Node filesystem:
    - `/proc`: Reads PSI metrics for OOM handling.
    - `/dev/watchdog`: Sends signals to reset the Watchdog timer.

The following external components interact with the module:

1. **Kube-apiserver**:
    - Executes mutating and validating webhooks of capi-controller-manager.
    - Forwards requests for bashible resources to bashible-api-server.
2. **Prometheus-main**:
    - Collects metrics from `node-manager` module components.

Architecture features specific to Static nodes

1. Users create and configure nodes in the following ways:
    - Manually, using bashible scripts preconfigured in DKP.
    - Manually, with the following node handover to CAPS for automated management.
    - Automatically, using CAPS.
2. **Capi-controller-manager** manages the lifecycle of the cluster and its nodes through higher-level custom resources, without directly provisioning infrastructure. It generates infrastructure-specific custom resources, leaving provisioning to the infrastructure provider, which is deployed by the specific cloud provider module (CAPS for static nodes).
3. **Caps-controller-manager**: Component managing static nodes (partially, without provisioning).
4. Static nodes can be used not only on bare metal, but in cloud as well. In that case, such a node is not managed by cloud-controller-manager, even if one of the cloud providers is enabled. Csi-driver is not installed on such nodes.
5. Automatic node scaling is not supported.

Additional resources

- Module `node-manager` documentation

Hybrid node group and cluster management

It is important to distinguish between hybrid node groups and hybrid clusters:

- **Hybrid node groups** may include both Static nodes deployed in the cloud and servers deployed in the customer's on-premises data center (bare metal or virtual machines).

For example, the primary workload may run on bare-metal servers, while cloud instances are used as a scalable addition during peak loads. Since nodes of the same type (Static) are used both in the customer's data center and in the cloud, the architecture of the `node-manager` module in this case corresponds to the Static node variant. The only mandatory requirement is the presence of L3 connectivity between the on-premises data center and the cloud. For example, Yandex Cloud provides a Cloud Interconnect mechanism.

- **Hybrid clusters** are DKP clusters that simultaneously include:
  - NodeGroup with Static nodes deployed in the customer's on-premises data center (bare metal or virtual machines).
  - NodeGroup with CloudEphemeral nodes deployed in the cloud.

  In this case, the components required to manage both node types are deployed. The architecture of the `node-manager` module corresponds to the Static node variant. In addition, components required for CloudEphemeral nodes are deployed:
  - **cloud-provider**: Provides interaction with the cloud infrastructure. A configured provider for the corresponding cloud is required. It also includes csi-driver and cloud-controller-manager.
  - **cluster-autoscaler**: Provides automatic scaling of cluster nodes.

  As with hybrid node groups, hybrid clusters require L3 connectivity between the on-premises data center and the cloud.

Additional resources

- Module `node-manager` documentation

Cloud-provider-aws module

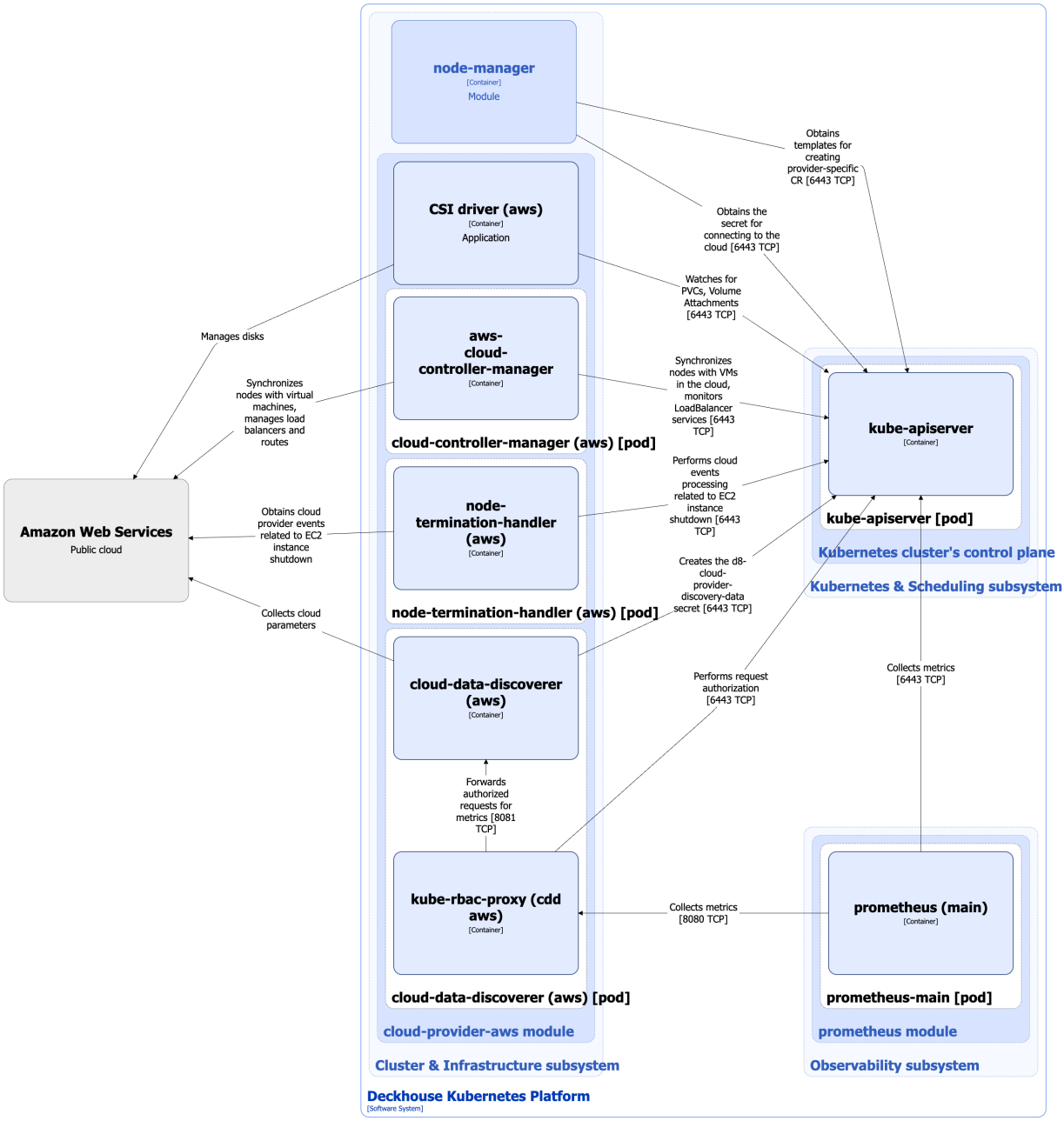
The `cloud-provider-aws` module is responsible for interacting with the Amazon Web Services cloud resources. It allows the `node-manager` module to use AWS resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-aws` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-controller-manager**: It is an implementation of cloud controller manager for AWS. It provides interaction with the AWS cloud and performs the following functions:
   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.
   - Creates network routes for the `PodNetwork` network in the cloud.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **aws-cloud-controller-manager**.

2. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-aws` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.

- **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

3. **CSI driver (aws)**: It is an implementation of the CSI driver for AWS. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation section.
4. **Node-termination-handler**: AWS Node Termination Handler. It is responsible for gracefully handling the termination of EC2 instances in the Kubernetes control plane.

   Node-termination-handler processes the following AWS events:

   - EC2 maintenance events
   - EC2 Spot interruptions
   - ASG Scale-In
   - ASG AZ Rebalance
   - EC2 Instance Termination

   It consists of a single container:

   - **node-termination-handler**.

   After receiving an event when it is necessary to drain and then to delete a node, for example, in case of a regular interruption of a spot instance, node-termination-handler sets the `aws-node-termination-handler/spot-itn` taint for the corresponding node. A node drain itself, and then node removal are performed by the `node-manager` module using the module hooks mechanics.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **Amazon Web Services**:
   - Collects cloud parameters.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages network routes for `PodNetwork` network.
   - Collects cloud events of EC2 instance termination.
   - Manages disks.

The following external components interact with the module:

- **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-aws` module provides `node-manager` with following artifacts:
   - Provider-specific custom resource templates to be used by `cloud-provider-aws` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific custom resources created based on the templates mentioned above.
2. The `cloud-provider-aws` module provides Terraform/OpenTofu components for AWS cloud used when building the `dhctl` executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty
     - `master-node`
     - `static-node`.

Additional resources

- Module `cloud-provider-aws` documentation
- Module `node-manager` documentation

- Module `terraform-manager` documentation

Cloud-provider-azure module

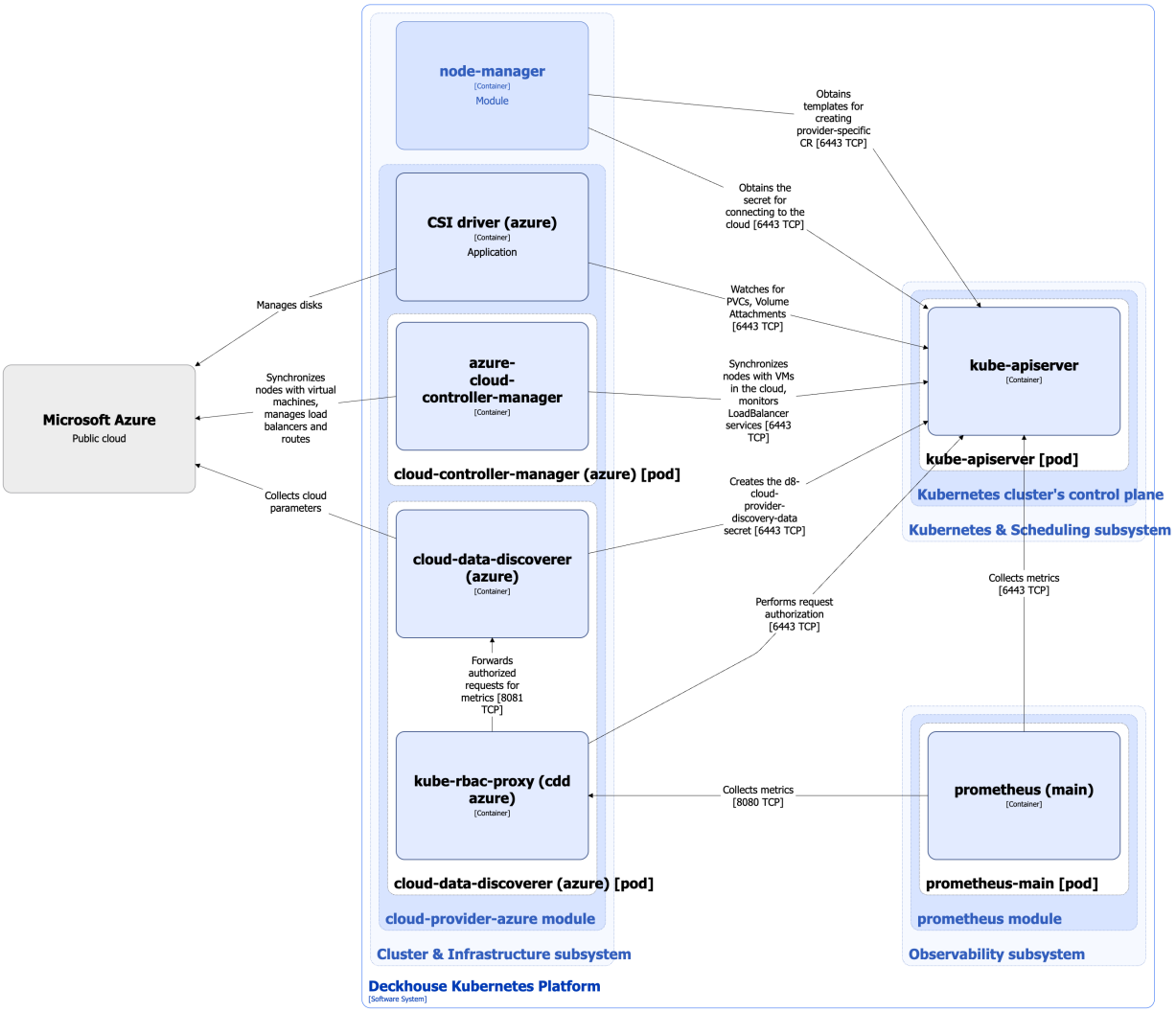
The `cloud-provider-azure` module is responsible for interacting with the Microsoft Azure cloud resources. It allows the `node-manager` module to use Microsoft Azure resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-azure` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-controller-manager**: It is an implementation of cloud controller manager for Microsoft Azure. It provides interaction with the Azure cloud and performs the following functions:
   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.

        - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
    - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.
    - Creates network routes for the `PodNetwork` network in the cloud.

  For more details about cloud-controller-manager, refer to the Kubernetes documentation.

  It consists of a single container:

    - **azure-cloud-controller-manager**.

2. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-azure` module.

   It consists of the following containers:

    - **cloud-data-discoverer**: Main container.
    - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

3. **CSI driver (azure)**: It is an implementation of the CSI driver for Microsoft Azure. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation section.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

    - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
    - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
    - Synchronizes Kubernetes nodes with cloud VMs.
    - Watches for LoadBalancer services.
    - Authorizes the requests for metrics.

2. **Microsoft Azure**:

    - Collects cloud parameters.
    - Gets `ProviderId` and other information about the VMs that are cluster nodes.
    - Manages load balancers.
    - Manages network routes for `PodNetwork` network.
    - Manages disks.

The following external components interact with the module:

- **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-azure` module provides `node-manager` with following artifacts:

    - Provider-specific custom resource templates to be used by `cloud-provider-azure` to create VMs in the cloud.
    - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific custom resources created based on the templates mentioned above.

2. The `cloud-provider-azure` module provides Terraform/OpenTofu components for Microsoft Azure cloud used when building the `dhctl` executable file for the `terraform-manager` module, such as:

    - Terraform/OpenTofu provider.
    - Terraform modules.

    - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:

        - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty
        - `master-node`
        - `static-node`.

Additional resources

- Module `cloud-provider-azure` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

Cloud-provider-dvp module

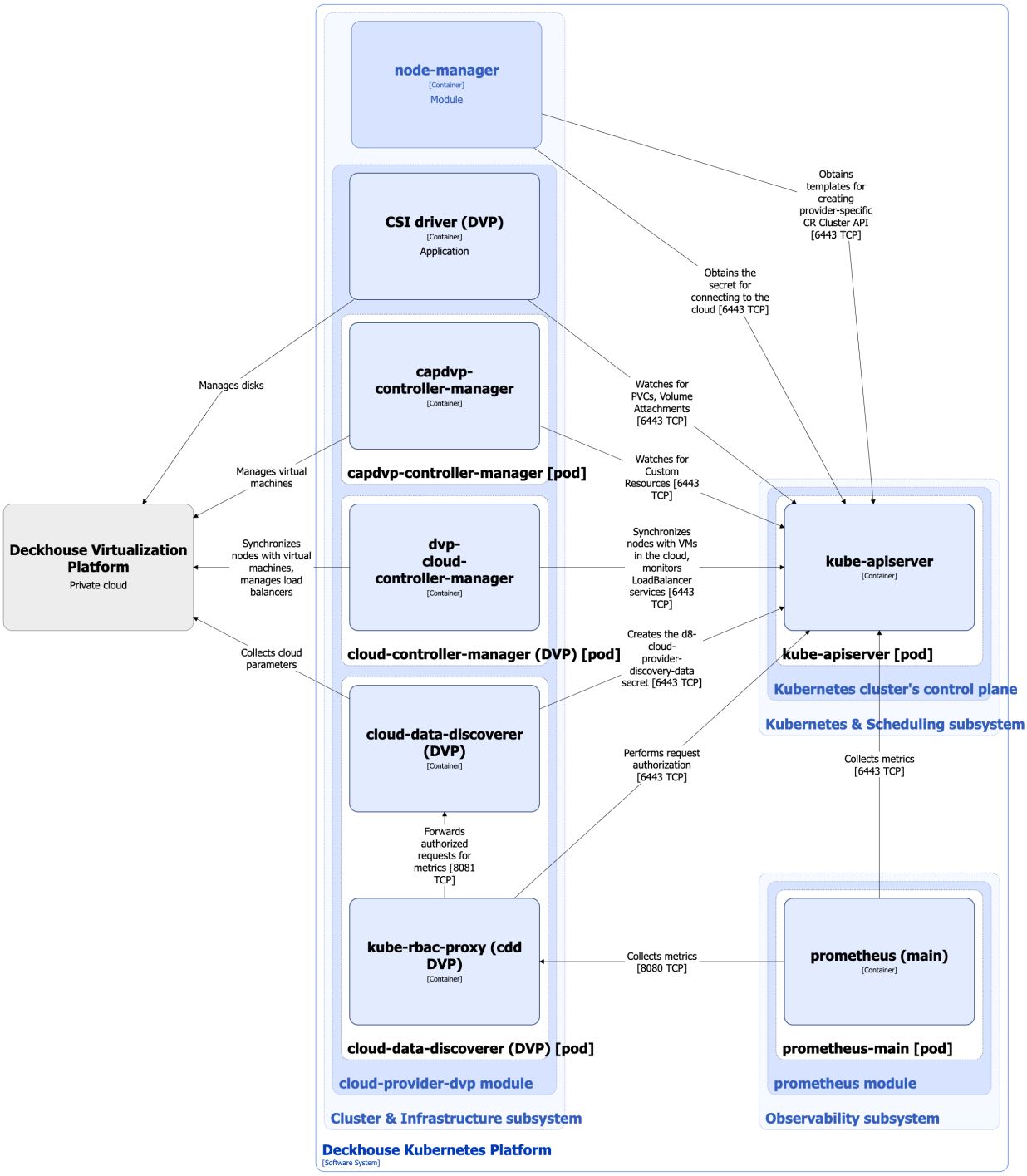
The `cloud-provider-dvp` module is responsible for interacting with the Deckhouse Virtualization Platform (DVP) cloud resources. It allows the `node-manager` module to use DVP resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-dvp` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Capdvp-controller-manager**: Kubernetes Cluster API Provider for DVP. Cluster API is an extension for Kubernetes that allows you to manage Kubernetes clusters as custom resources inside another Kubernetes cluster. Cluster API Provider allows clusters running the Cluster API to order virtual machines in the cloud provider's infrastructure, DVP in this case. Capdvp-controller-manager works with the following custom resources:
   - DeckhouseCluster: Description of a DVP-based cluster.
   - DeckhouseMachineTemplate: Template describing the characteristics of the machines created in the cloud.
   - DeckhouseMachine: Description of the characteristics of a machine created based on DeckhouseMachineTemplate.

   It consists of a single container:
   - **capdvp-controller-manager**.
2. **Cloud-controller-manager**: It is an implementation of Cloud controller manager for DVP. It provides interaction with the DVP cloud and performs the following functions:
   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:
   - **dvp-cloud-controller-manager**.
3. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-dvp` module. For example, for DVP, these are parameters such as a list of availability zones, StorageClass resources, etc.

   It consists of the following containers:
   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.
4. **CSI driver (DVP)**: It is an implementation of the CSI driver for DVP. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation page.

   CSI driver (DVP) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Reconciles DeckhouseCluster, DeckhouseMachineTemplate, and DeckhouseMachine custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **DVP cloud**:
   - Collects cloud parameters.
   - Manages virtual machines.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.

The following external components interact with the module:

1. **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-dvp` module provides `node-manager` with following artifacts:

    - Provider-specific Cluster API custom resource templates to be used by `cloud-provider-dvp` to create VMs in the cloud.
    - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific Cluster API custom resources created based on the templates mentioned above.

2. The `cloud-provider-dvp` module provides Terraform/OpenTofu components for DVP cloud used when building the dhctl executable file for the `terraform-manager` module, such as:

    - Terraform/OpenTofu provider.
    - Terraform modules.

    - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:

        - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty.
        - `master-node`
        - `static-node`

Additional resources

- Module `cloud-provider-dvp` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

Cloud-provider-dynamix module

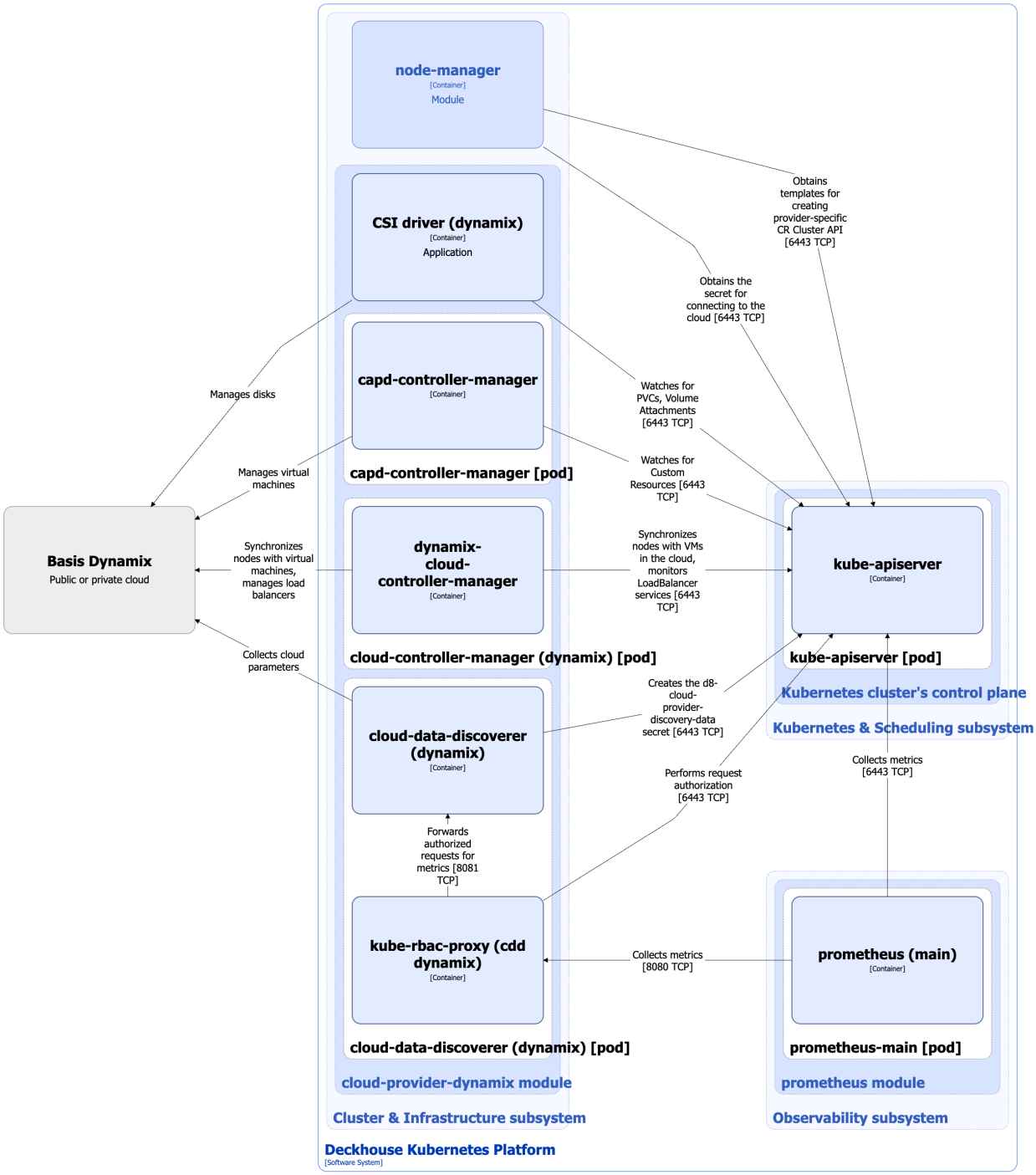
The `cloud-provider-dynamix` module is responsible for interacting with the Basis Dynamix cloud resources. It allows the `node-manager` module to use Basis Dynamix resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-dynamix` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Capd-controller-manager**: Kubernetes Cluster API Provider for Basis Dynamix. Cluster API is an extension for Kubernetes that allows you to manage Kubernetes clusters as custom resources inside another Kubernetes cluster. Cluster API Provider allows clusters running the Cluster API to order virtual machines in the cloud provider's infrastructure, Basis Dynamix in this case. Capd-controller-manager works with the following custom resources:

   - DynamixCluster: Description of a Basis Dynamix-based cluster.
   - DynamixMachineTemplate: Template describing the characteristics of the machines created in the cloud.
   - DynamixMachine: Description of the characteristics of a machine created based on DynamixMachineTemplate.

   It consists of a single container:

   - **capd-controller-manager**.

2. **Cloud-controller-manager**: It is an implementation of Cloud controller manager for Basis Dynamix. It provides interaction with the Basis Dynamix cloud and performs the following functions:

   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider.

To do this:

- It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
- It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.

- When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

For more details about cloud-controller-manager, refer to the Kubernetes documentation.

It consists of a single container:

- **dynamix-cloud-controller-manager**.

3. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-dynamix` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

4. **CSI driver (dynamix)**: It is an implementation of the CSI driver for Basis Dynamix. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation page.

   CSI driver (dynamix) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Reconciles DynamixCluster, DynamixMachineTemplate, and DynamixMachine custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **Basis Dynamix cloud**:
   - Collects cloud parameters.
   - Manages virtual machines.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.

The following external components interact with the module:

1. **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-dynamix` module provides `node-manager` with following artifacts:
   - Provider-specific Cluster API custom resource templates to be used by `cloud-provider-dynamix` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific Cluster API custom resources created based on the templates mentioned above.
2. The `cloud-provider-dynamix` module provides Terraform/OpenTofu components for Basis Dynamix cloud used when building the dhctl executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty.
     - `master-node`
     - `static-node`

Additional resources

- Module `cloud-provider-dynamix` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

Cloud-provider-gcp module

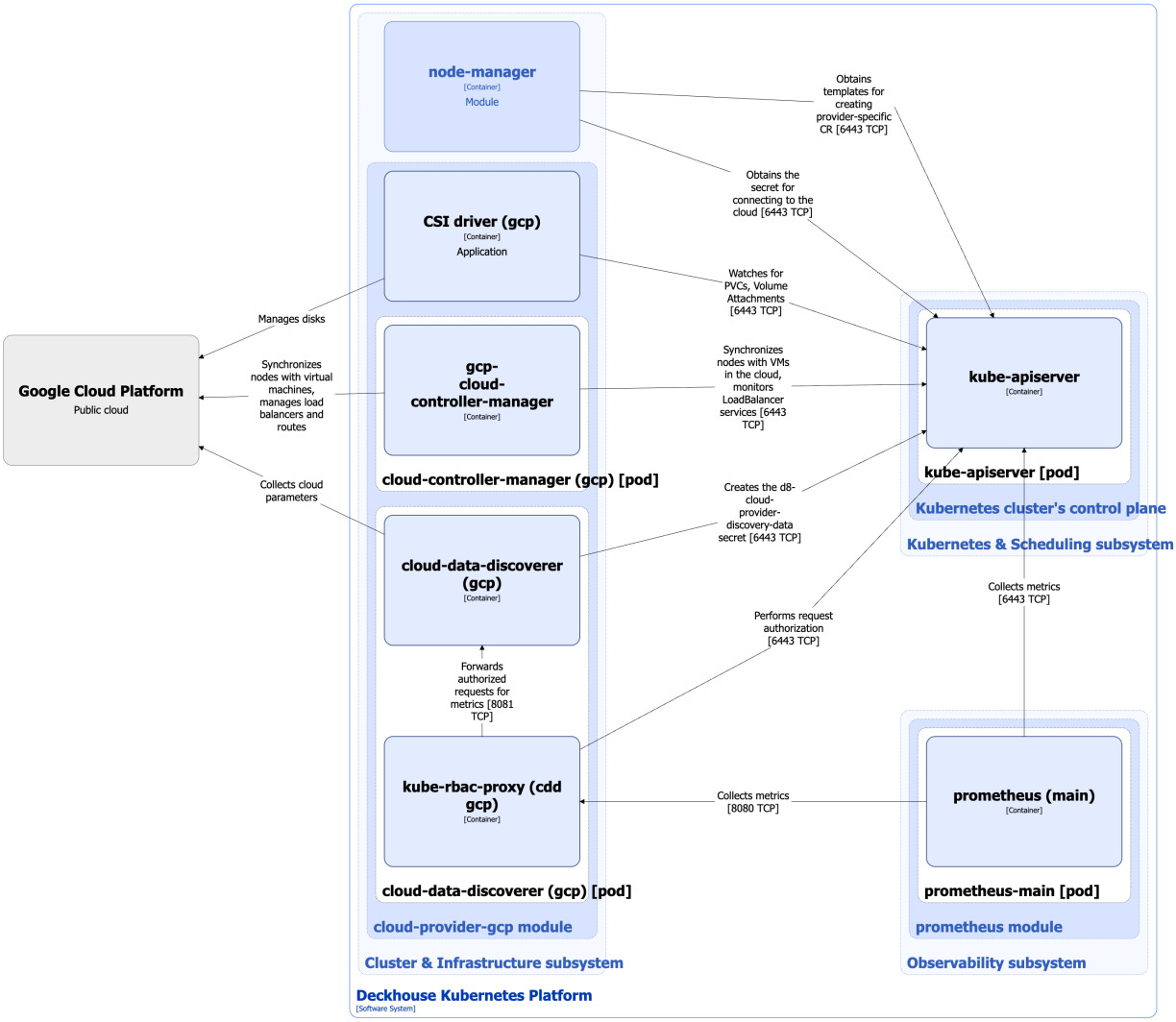
The `cloud-provider-gcp` module is responsible for interacting with the Google Cloud Platform cloud resources. It allows the `node-manager` module to use GCP resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-gcp` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-controller-manager**: It is an implementation of cloud controller manager for GCP. It provides interaction with the GCP cloud and performs the following functions:

- Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
  - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
  - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
- When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.
- Creates network routes for the `PodNetwork` network in the cloud.

For more details about cloud-controller-manager, refer to the Kubernetes documentation.

It consists of a single container:

- **gcp-cloud-controller-manager**.

2. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-gcp` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

3. **CSI driver (gcp)**: It is an implementation of the CSI driver for GCP. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation section.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **Google Cloud Platform**:
   - Collects cloud parameters.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages network routes for `PodNetwork` network.
   - Manages disks.

The following external components interact with the module:

- **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-gcp` module provides `node-manager` with following artifacts:
   - Provider-specific custom resource templates to be used by `cloud-provider-gcp` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific custom resources created based on the templates mentioned above.
2. The `cloud-provider-gcp` module provides Terraform/OpenTofu components for GCP cloud used when building the `dhctl` executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty
     - `master-node`
     - `static-node`.

Additional resources

- Module `cloud-provider-gcp` documentation

- Module `node-manager` documentation
- Module `terraform-manager` documentation

Cloud-provider-huaweicloud module

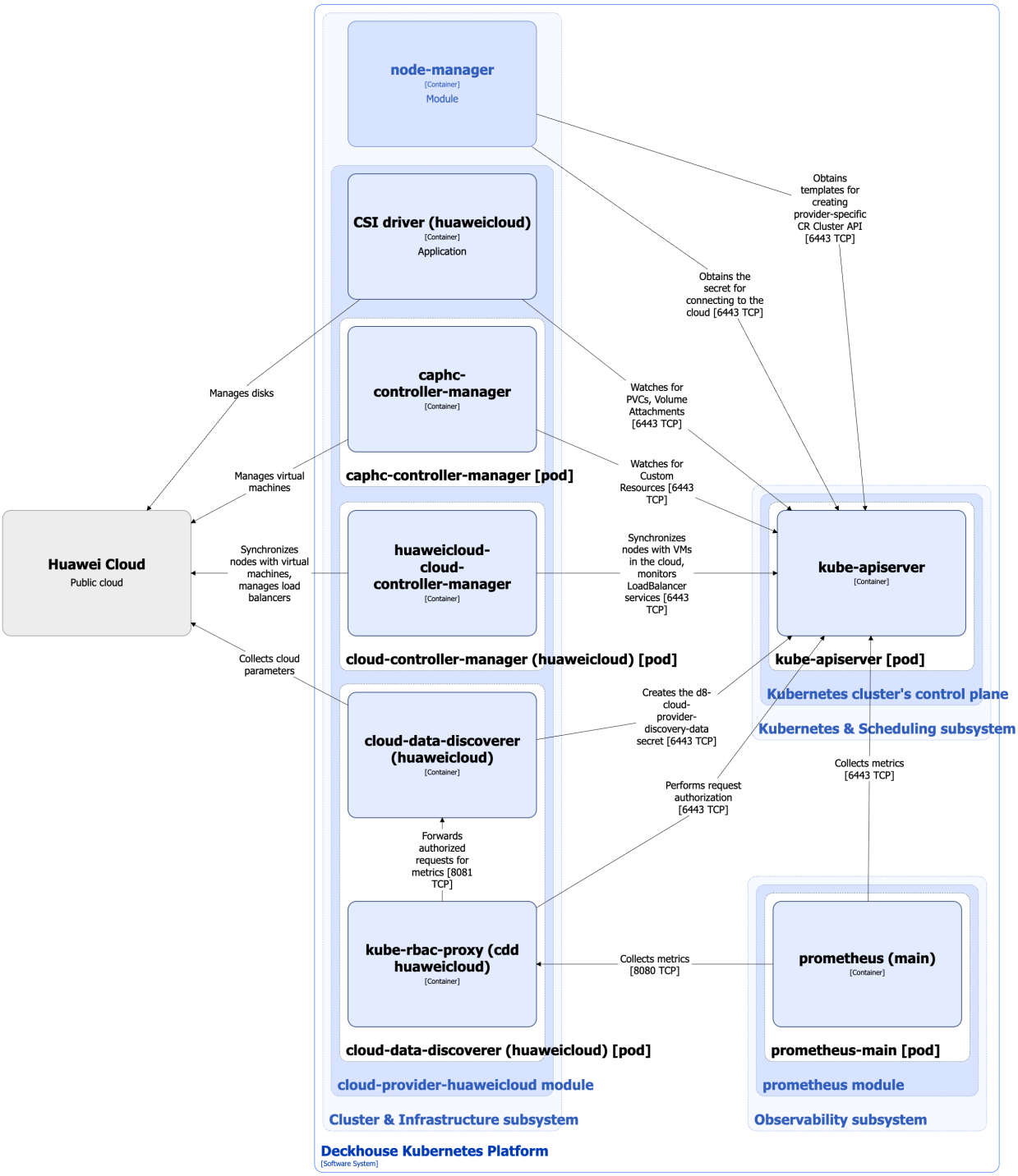
The `cloud-provider-huaweicloud` module is responsible for interacting with the Huawei Cloud resources. It allows the `node-manager` module to use Huawei Cloud resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-huaweicloud` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Caphc-controller-manager**: Kubernetes Cluster API Provider for Huawei Cloud. Cluster API is an extension for Kubernetes that allows you to manage Kubernetes clusters as custom resources inside another Kubernetes cluster. Cluster API Provider allows clusters running the Cluster API to order virtual machines in the cloud provider’s infrastructure, Huawei Cloud in this case. Caphc-controller-manager works with the following custom resources:

   - HuaweiCloudCluster: Description of a Huawei Cloud-based cluster.
   - HuaweiCloudMachineTemplate: Template describing the characteristics of the machines created in the cloud.
   - HuaweiCloudMachine: Description of the characteristics of a machine created based on HuaweiCloudMachineTemplate.

   It consists of a single container:

   - **caphc-controller-manager**.

2. **Cloud-controller-manager**: It is an implementation of Cloud controller manager for Huawei Cloud. It provides interaction with the Huawei Cloud and performs the following functions:

- Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
  - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
  - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
- When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

For more details about cloud-controller-manager, refer to the Kubernetes documentation.

It consists of a single container:

- **huaweicloud-cloud-controller-manager**.

3. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-huaweicloud` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

4. **CSI driver (huaweicloud)**: It is an implementation of the CSI driver for Huawei Cloud. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation page.

   CSI driver (huaweicloud) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Reconciles HuaweiCloudCluster, HuaweiCloudMachineTemplate, and HuaweiCloudMachine custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **Huawei Cloud**:
   - Collects cloud parameters.
   - Manages virtual machines.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.

The following external components interact with the module:

1. **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-huaweicloud` module provides `node-manager` with following artifacts:
   - Provider-specific Cluster API custom resource templates to be used by `cloud-provider-huaweicloud` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific Cluster API custom resources created based on the templates mentioned above.
2. The `cloud-provider-huaweicloud` module provides Terraform/OpenTofu components for Huawei Cloud used when building the dhctl executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty.
     - `master-node`

    - `static-node`

Additional resources

- Module `cloud-provider-huaweicloud` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

Cloud-provider-openstack module

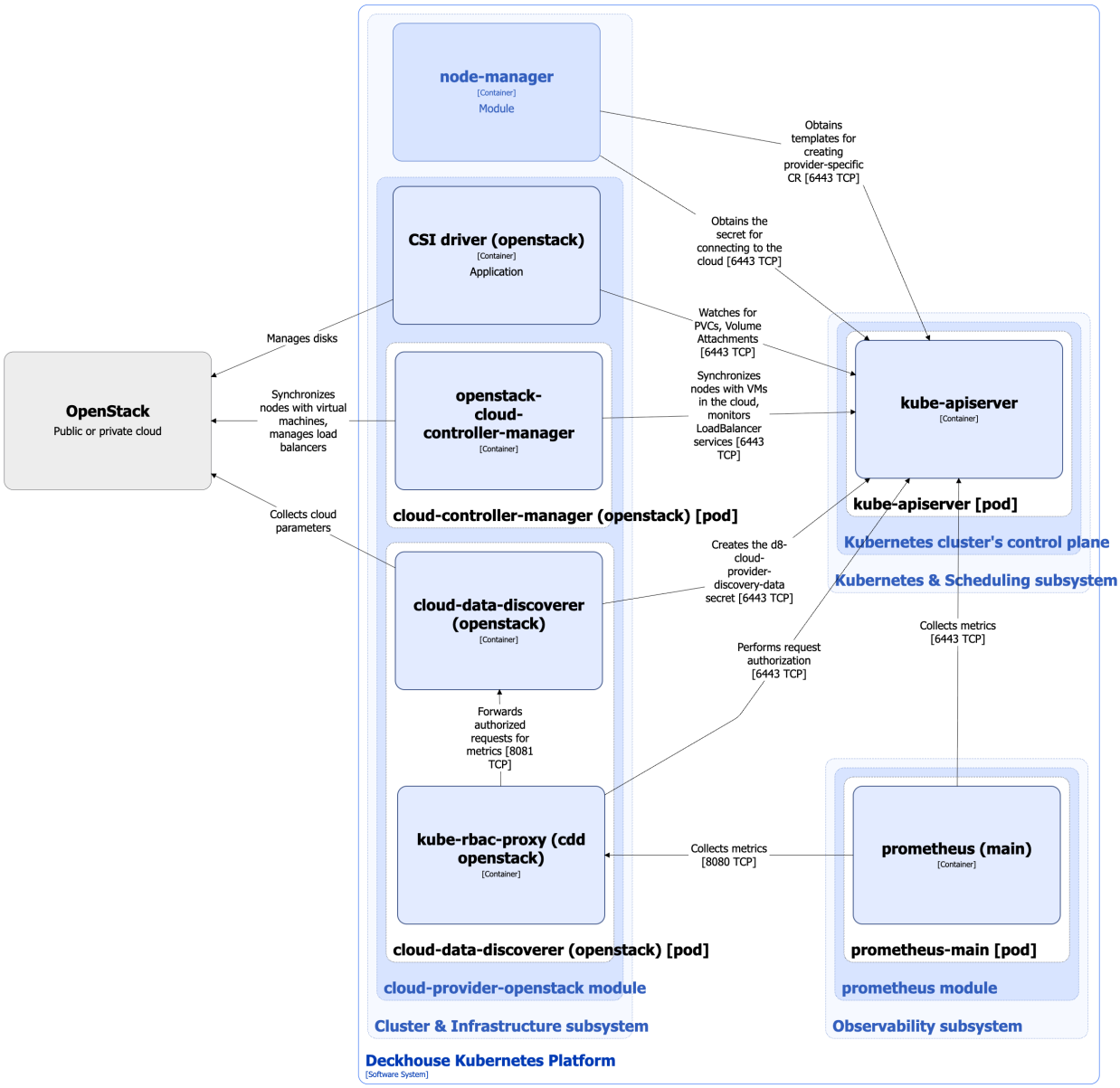
The `cloud-provider-openstack` module is responsible for interacting with the OpenStack cloud resources. It allows the `node-manager` module to use OpenStack resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-openstack` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-controller-manager**: Kubernetes external cloud provider for OpenStack , an implementation of Cloud controller manager for OpenStack. It provides interaction with the OpenStack cloud and performs the following functions:
   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.
   - Creates network routes for the `PodNetwork` network in the cloud.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **openstack-cloud-controller-manager**.

2. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-openstack` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

3. **CSI driver (openstack)**: It is an implementation of the CSI driver for OpenStack. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation page.

   CSI driver (openstack) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **OpenStack**:
   - Collects cloud parameters.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages network routes for `PodNetwork` network.
   - Manages disks.

The following external components interact with the module:

1. **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-openstack` module provides `node-manager` with following artifacts:
   - Provider-specific custom resource templates to be used by `cloud-provider-openstack` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific custom resources created based on the templates mentioned above.
2. The `cloud-provider-openstack` module provides Terraform/OpenTofu components for OpenStack cloud used when building the dhctl executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty.
     - `master-node`
     - `static-node`

Additional resources

- Module `cloud-provider-openstack` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

Cloud-provider-vcd module

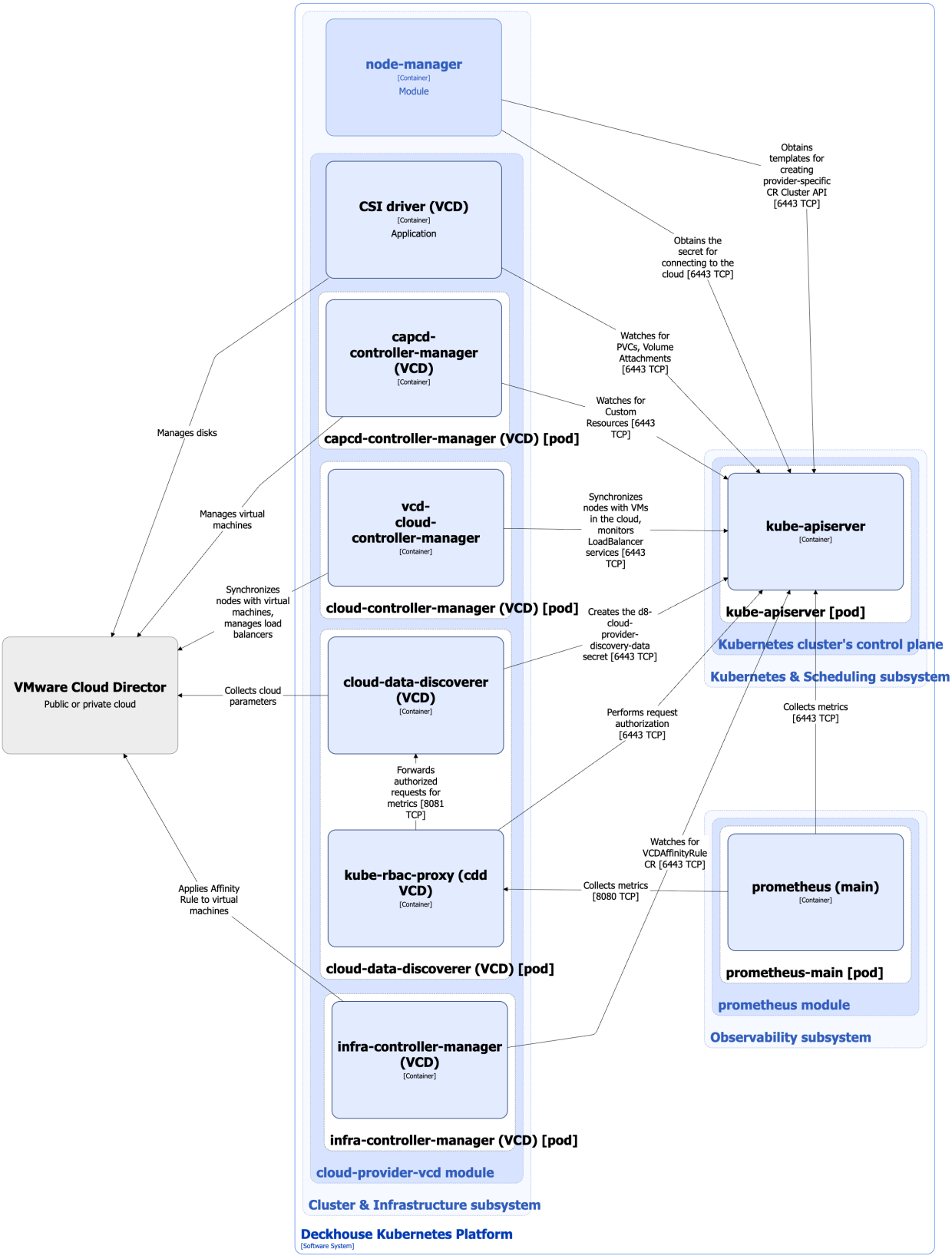
The `cloud-provider-vcd` module is responsible for interacting with the VMware Cloud Director cloud resources. It allows the `node-manager` module to use VMware Cloud Director resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-vcd` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Capcd-controller-manager**: Kubernetes Cluster API Provider Cloud Director. Cluster API is an extension for Kubernetes that allows you to manage Kubernetes clusters as custom resources inside another Kubernetes cluster. Cluster API Provider allows clusters running the Cluster API to order virtual machines in the cloud provider's infrastructure, VMware Cloud Director in this case. Capcd-controller-manager works with the following custom resources:
   - VCDClusterTemplate: Template describing the characteristics of the cluster created in the cloud.
   - VCDCluster: Description of the characteristics of a cluster created based on VCDClusterTemplate.
   - VCDMachineTemplate: Template describing the characteristics of the machines created in the cloud.
   - VCDMachine: Description of the characteristics of a machine created based on VCDMachineTemplate.

   It consists of a single container:

   - **capcd-controller-manager**.
2. **Cloud-controller-manager**: Kubernetes External Cloud Provider for VMware Cloud Director. It is an implementation of cloud controller manager for VMware Cloud Director. It provides interaction with the VMware Cloud Director cloud and performs the following functions:
   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **vcd-cloud-controller-manager**.
3. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-vcd` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.
4. **Infra-controller-manager**: It is responsible for managing the schedulling of node location relative to each other at the hypervisor level, which can help improve fault tolerance, and control the distribution of workloads. Infra-controller-manager works with a custom resource VCDAffinityRule. It specifies the affinity rule applied only to virtual machines of node groups referencing this instance class within the cluster.

   It consists of a single container:

   - **infra-controller-manager**.
5. **CSI driver (VCD)**: It is an implementation of the CSI driver for VMware Cloud Director. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation section. `Cloud-provider-vcd` module uses CSI driver for VMware Cloud Director Named Independent Disks.

   CSI driver (VCD) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Reconciles VCDClusterTemplate, VCDCluster, VCDMachineTemplate, VCDMachine, and VCDAffinityRule custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **VMware Cloud Director**:
   - Collects cloud parameters.
   - Manages virtual machines.
   - Applies Affinity Rules to virtual machines.

    - Gets `ProviderId` and other information about the VMs that are cluster nodes.
    - Manages load balancers.
    - Manages disks.

The following external components interact with the module:

- **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-vcd` module provides `node-manager` with following artifacts:
    - Provider-specific Cluster API custom resource templates to be used by `cloud-provider-vcd` to create VMs in the cloud.
    - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific Cluster API custom resources created based on the templates mentioned above.
2. The `cloud-provider-vcd` module provides Terraform/OpenTofu components for VMware Cloud Director cloud used when building the `dhctl` executable file for the `terraform-manager` module, such as:
    - Terraform/OpenTofu provider.
    - Terraform modules.
    - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
        - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty
        - `master-node`
        - `static-node`.

Additional resources

- Module `cloud-provider-vcd` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

Cloud-provider-vsphere module

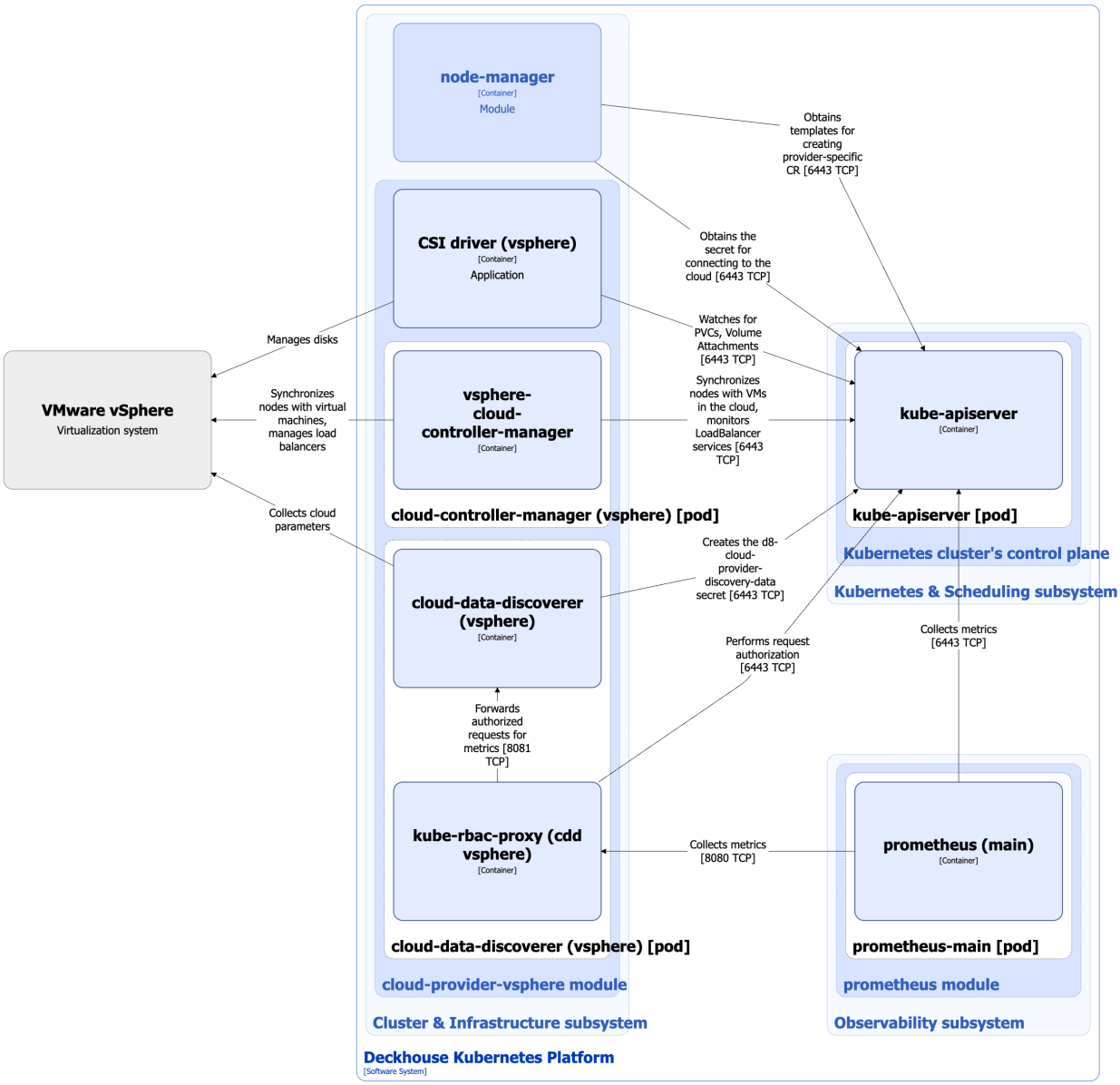
The `cloud-provider-vsphere` module is responsible for interacting with the VMware vSphere cloud resources. It allows the `node-manager` module to use VMware vSphere resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-vsphere` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-controller-manager**: Kubernetes vSphere Cloud Provider. It is an implementation of cloud controller manager for VMware vSphere. It provides interaction with the VMware vSphere cloud and performs the following functions:

   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **vsphere-cloud-controller-manager**.

2. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-vsphere` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

3. **CSI driver (vsphere)**: It is an implementation of the CSI driver for VMware vSphere. To study the `cloud-provider-vsphere` CSI driver architecture, refer to the corresponding documentation section.

   CSI driver (vsphere) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **VMware vSphere**:
   - Collects cloud parameters.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.

The following external components interact with the module:

- **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-vsphere` module provides `node-manager` with following artifacts:
   - Provider-specific custom resource templates to be used by `cloud-provider-vsphere` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific custom resources created based on the templates mentioned above.
2. The `cloud-provider-vsphere` module provides Terraform/OpenTofu components for VMware vSphere cloud used when building the `dhctl` executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty
     - `master-node`
     - `static-node`.

Additional resources

- Module `cloud-provider-vsphere` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

Cloud-provider-yandex module

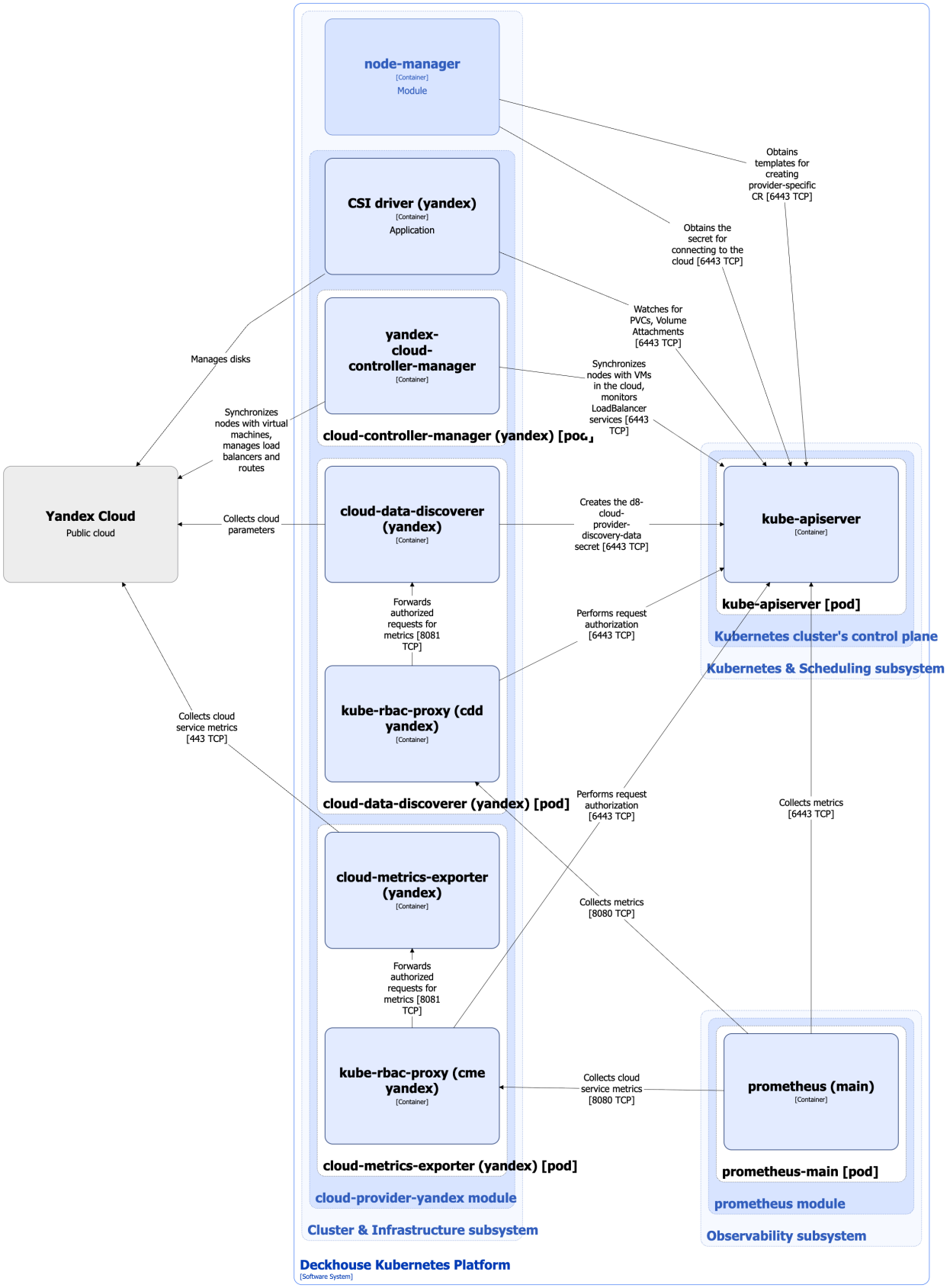
The `cloud-provider-yandex` module is responsible for interacting with the Yandex Cloud cloud resources. It allows the `node-manager` module to use Yandex Cloud resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-yandex` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-controller-manager**: It is an implementation of Cloud controller manager for Yandex Cloud. It provides interaction with the Yandex Cloud and performs the following functions:
   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider. To do this:
     - It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
     - It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.
   - When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.
   - Creates network routes for the `PodNetwork` network in the cloud.

   For more details about cloud-controller-manager, refer to the Kubernetes documentation.

   It consists of a single container:

   - **yandex-cloud-controller-manager**.

2. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-yandex` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

3. **Cloud-metrics-exporter**: It is responsible for collecting metrics from the cloud provider's API and providing them in Prometheus format.

   It consists of the following containers:

   - **cloud-metrics-exporter**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-metrics-exporter metrics.

4. **CSI driver (yandex)**: It is an implementation of the CSI driver for Yandex Cloud. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation page.

   CSI driver (yandex) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.

2. **Yandex Cloud**:
   - Collects cloud parameters.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.
   - Manages network routes for `PodNetwork` network.
   - Collects cloud service metrics.

The following external components interact with the module:

1. **Prometheus-main**:
   - Collects cloud-data-discoverer metrics.
   - Collects cloud provider's metrics using cloud-metrics-exporter.

Indirect interactions:

1. The `cloud-provider-yandex` module provides `node-manager` with following artifacts:
   - Provider-specific custom resource templates to be used by `cloud-provider-yandex` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary

settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific custom resources created based on the templates mentioned above.

2. The `cloud-provider-yandex` module provides Terraform/OpenTofu components for Yandex Cloud used when building the dhctl executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty.
     - `master-node`
     - `static-node`

Additional resources

- Module `cloud-provider-yandex` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

Cloud-provider-zvirt module

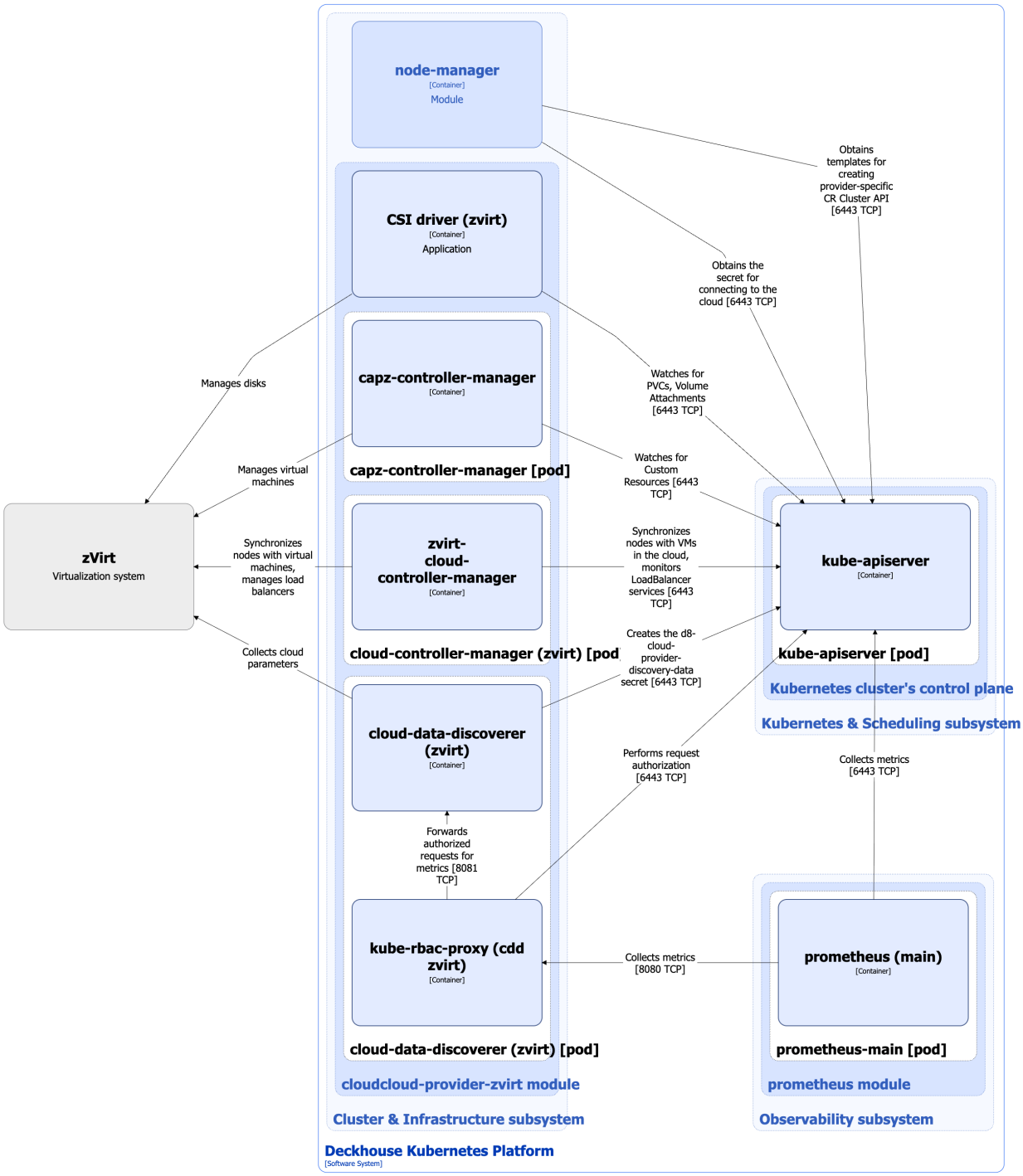
The `cloud-provider-zvirt` module is responsible for interacting with the zVirt cloud resources. It allows the `node-manager` module to use zVirt resources for provisioning nodes for the specified node group.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cloud-provider-zvirt` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Capz-controller-manager**: Kubernetes Cluster API Provider for zVirt. Cluster API is an extension for Kubernetes that allows you to manage Kubernetes clusters as custom resources inside another Kubernetes cluster. Cluster API Provider allows clusters running the Cluster API to order virtual machines in the cloud provider’s infrastructure, zVirt in this case. Capz-controller-manager works with the following custom resources:

   - ZvirtCluster: Description of a zVirt-based cluster.
   - ZvirtMachineTemplate: Template describing the characteristics of the machines created in the cloud.
   - ZvirtMachine: Description of the characteristics of a machine created based on ZvirtMachineTemplate.

   It consists of a single container:

   - **capz-controller-manager**.

2. **Cloud-controller-manager**: It is an implementation of Cloud controller manager for zVirt. It provides interaction with the zVirt cloud and performs the following functions:

   - Implements a 1:1 relationship between a Node resource in Kubernetes and a VM in a cloud provider.

To do this:

- It fills the `spec.providerId` and `NodeInfo` fields of the Node resource.
- It checks for a VM in the cloud and deletes the Node resource in the cluster if it is missing.

- When creating a LoadBalancer Service resource in Kubernetes, it creates a load balancer in the cloud that routes traffic from outside into the cluster nodes.

For more details about cloud-controller-manager, refer to the Kubernetes documentation.

It consists of a single container:

- **zvirt-cloud-controller-manager**.

3. **Cloud-data-discoverer**: It is responsible for collecting data from the cloud provider's API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used by other components of the `cloud-provider-zvirt` module.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

4. **CSI driver (zvirt)**: It is an implementation of the CSI driver for zVirt. To study the `cloud-provider-*` CSI driver typical architecture, refer to the corresponding documentation page.

   CSI driver (zvirt) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Reconciles ZvirtCluster, ZvirtMachineTemplate, and ZvirtMachine custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Synchronizes Kubernetes nodes with cloud VMs.
   - Watches for LoadBalancer services.
   - Authorizes the requests for metrics.
2. **zVirt**:
   - Collects cloud parameters.
   - Manages virtual machines.
   - Gets `ProviderId` and other information about the VMs that are cluster nodes.
   - Manages load balancers.
   - Manages disks.

The following external components interact with the module:

1. **Prometheus-main**: Collects cloud-data-discoverer metrics.

Indirect interactions:

1. The `cloud-provider-zvirt` module provides `node-manager` with following artifacts:
   - Provider-specific Cluster API custom resource templates to be used by `cloud-provider-zvirt` to create VMs in the cloud.
   - The `kube-system/d8-node-manager-cloud-provider` Secret, which contains all the necessary settings to connect to the cloud and to create CloudEphemeral nodes. These settings are registered in the provider-specific Cluster API custom resources created based on the templates mentioned above.
2. The `cloud-provider-zvirt` module provides Terraform/OpenTofu components for zVirt cloud used when building the dhctl executable file for the `terraform-manager` module, such as:
   - Terraform/OpenTofu provider.
   - Terraform modules.
   - Layouts: Set of cloud placement schemes, which define how the basic infrastructure is created, how and with which additional characteristics should nodes be created for this placement. For example, for one scheme, nodes may have public IP addresses, but they will not for the other. Each layout should have three modules:
     - `base-infrastructure`: Basic infrastructure (for example, creation of networks), can also be empty.
     - `master-node`
     - `static-node`

Additional resources

- Module `cloud-provider-zvirt` documentation
- Module `node-manager` documentation
- Module `terraform-manager` documentation

Chrony module

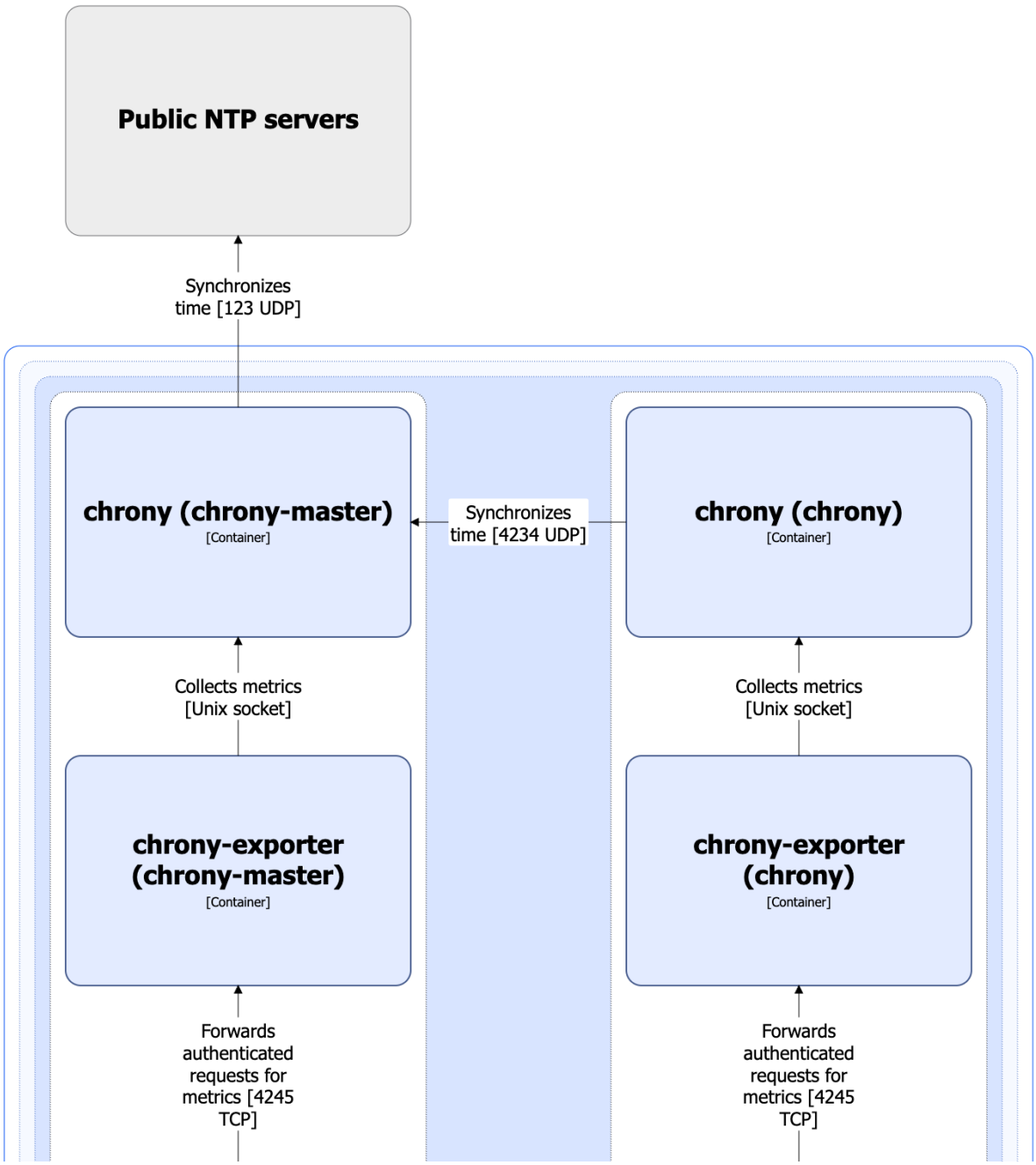
The `chrony` module provides time synchronization on all nodes in the Deckhouse Kubernetes Platform (DKP) cluster using chrony NTP server/client implementation.

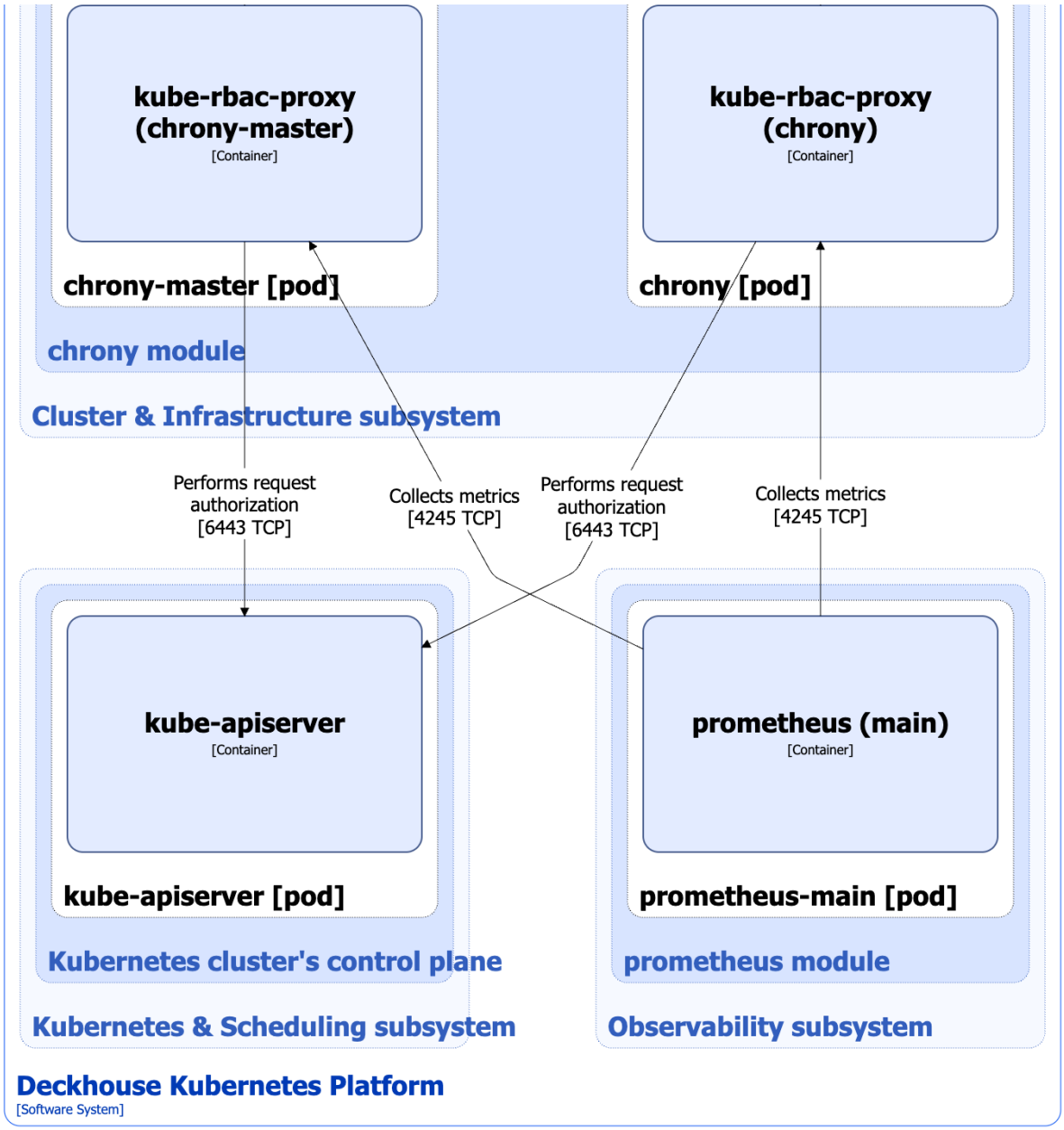
For more details, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `chrony` module its interactions with other DKP components are shown in the following diagram:





Module components

The module consists of the following components:

1. **Chrony-master**: Time synchronization service on cluster master nodes with external NTP servers.

   It consists of the following containers:

   - **chrony**: Main container.
   - **chrony-exporter**: A sidecar container that collects metrics from the chrony container and exposes them in Prometheus format.
   - **kube-rbac-proxy**: A sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to metrics from the chrony-exporter container.

2. **Chrony**: Time synchronization service on all cluster nodes except master nodes, synchronized with the chrony-master component.

   It consists of the following containers:

   - **chrony**: Main container.
   - **chrony-exporter**: A sidecar container that collects metrics from the chrony container and exposes them in Prometheus format.
   - **kube-rbac-proxy**: A sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to metrics from the chrony-exporter container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**: Authorizes requests to retrieve metrics.
2. **External NTP servers**: Performs time synchronization.

The following external components interact with the module:

- **Prometheus-main**: Collects metrics from chrony and chrony-master.

Additional resources

- Module `chrony` documentation

Gpu module

The `gpu` module manages Graphics Processing Units (GPU) in Deckhouse Kubernetes Platform (DKP).

The module operates in two mutually exclusive modes. The mode is defined by the `dra.enabled` parameter:

- Dynamic Resource Allocation (DRA): The module uses a Kubernetes mechanism for requesting and sharing devices that provides dynamic and declarative allocation of GPU compute resources.
- Device Plugin mode (default): The module uses the classic Kubernetes model for working with node compute resources and publishes `nvidia.com/gpu` or `nvidia.com/mig-*` resources, which kube-scheduler uses to schedule pods that consume these resources.

For more details, refer to the corresponding documentation section.

The module architecture depends on the operating mode.

DRA mode

In DRA mode, the module is built around a vendor-independent core and adapters for the NVIDIA and MetaX vendors.

The module works with the following resources:

- DeviceClass: A DRA resource that stores descriptions of device classes that can be used for dynamic resource assignment.
- GPUClass: A custom resource that stores requirements for a GPU group (memory size, hardware capabilities) and the usage and compatibility policy.
- PhysicalGPU: A custom resource that stores a description of a physical GPU, including device characteristics.
- ResourceClaim: A DRA resource that contains a request to allocate a resource for a pod and describes the required characteristics and usage parameters.
- ResourceSlice: A DRA resource that represents an allocated share or part of a resource assigned within a ResourceClaim.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `gpu` module in DRA mode and its interactions with other DKP components are shown in the following diagram:

Module components

The module consists of the following components:

1. **gpu-controller** (Deployment): A controller that processes GPU resource requests and runs admission webhooks for DRA objects through the Validating/Mutating Admission Controllers mechanism. The controller runs on master nodes.

   The gpu-controller performs the following actions:

   - Creates and updates DeviceClass DRA resources based on PhysicalGPU and GPUClass custom resources.
   - Validates and mutates Pod create and update requests, creating ResourceClaim resources when necessary.
   - Watches ResourceClaim resource changes and, based on this data, maintains the state and occupancy of PhysicalGPU resources.

- Validates Pod, GPUClass, ResourceClaim, and DeviceClass resources.
- Manages the PhysicalGPU state.

It consists of the following containers:

- **gpu-controller**: Main container.
- **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the main container. This component is an Open Source project.

2. **gpu-node-agent** (DaemonSet): A component that consists of a single **gpu-node-agent** container and performs the following actions:

   - Scans the host `/sys` filesystem and the PCI ID database.
   - Matches devices against the `gpu-supported-vendors` ConfigMap.
   - Creates PhysicalGPU custom resources for each discovered GPU.
   - Sets the `gpu.deckhouse.io/vendor=<VENDOR>` labels on the Node resource.

   The component runs on all cluster nodes except the control plane.

3. **<VENDOR>-adapter** (DaemonSet): A component that works with hardware to prepare, allocate, and release GPU resources. Currently, two hardware vendors are supported: NVIDIA and MetaX.

   The component performs the following actions:

   - Registers in kubelet as a DRA kubelet plugin.
   - Prepares and releases allocated resources for pods through the `PrepareResourceClaims` and `UnprepareResourceClaims` operations.
   - Publishes the list of available devices through ResourceSlice resources.
   - Retrieves hardware capabilities.
   - Partitions GPU resources and makes them available to pods.
   - Enriches the status in PhysicalGPU resources.

   It consists of the following containers:

   - **dra-plugin**: Sidecar container that implements the DRA kubelet plugin and interacts with the vendor adapter.
   - **<VENDOR>-adapter**: Vendor-specific sidecar container that interacts with the hardware on the node.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the dra-plugin container.

   The component runs on all cluster nodes that have the `gpu.deckhouse.io/vendor=<VENDOR>` label.

4. **gpu-dcgm** (DaemonSet): A component that consists of a single **dcgm** container and runs Data Center GPU Manager (DCGM). DCGM collects GPU health and utilization data (Error Correction Code (ECC), power). It works only with NVIDIA cards.

5. **gpu-dcgm-exporter** (DaemonSet): A component that consists of a single **dcgm-exporter** container, retrieves GPU metrics from the gpu-dcgm component, and exposes them in Prometheus format.

6. **vfio-switch-<NODE_NAME>-<PCI>** (Job): A component that consists of a single **switch** container. It switches the driver in use between `nvidia` and `vfio-pci`. The component is created by nvidia-adapter to control the switching process.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Authorizes requests to retrieve metrics.
   - Works with PhysicalGPU and GPUClass custom resources.
   - Updates Node resources.
   - Validates Pod, GPUClass, ResourceClaim, and DeviceClass resources.
   - Works with DeviceClass, ResourceClaim, and ResourceSlice resources.
   - Creates and monitors `vfio-switch-<NODE_NAME>-<PCI>` Jobs.

2. **Kubelet**: Registers the module as a DRA kubelet plugin.

The following external components interact with the module:

1. **Kubelet**: Calls the `PrepareResourceClaims` and `UnprepareResourceClaims` gRPC methods.
2. **Kube-apiserver**: Validates Pod, GPUClass, ResourceClaim, and DeviceClass resources.
3. **Prometheus-main**: Collects metrics from the gpu-dcgm and <VENDOR>-adapter components.

Device Plugin mode

In Device Plugin mode, the module consists of components that work only with NVIDIA adapters.

The module interacts with the following resources:

- NodeFeature: Stores actual information about the hardware capabilities of a specific node.
- NodeFeatureRule: Stores a set of rules used by the module to configure labels, annotations, and taints for a cluster node.

Module architecture

The Level 2 C4 architecture of the `gpu` module in Device Plugin mode and its interactions with other DKP components are shown in the following diagram:

Module components

The module consists of the following components:

1. **node-feature-discovery-master** (Deployment): A component that consists of a single **master** container, which collects information about node hardware capabilities from NodeFeature resources, and publishes them as `feature.node.kubernetes.io/*` and `nvidia.com/*` labels on the corresponding nodes. The master obtains the rules for assigning labels, taints, and annotations from NodeFeatureRule resources.
2. **node-feature-discovery-worker** (DaemonSet): A component that consists of a single **worker** container, which runs on each GPU node, discovers connected PCI and USB devices on the node, and publishes them as NodeFeature resources. The component also publishes information received from the gpu-feature-discovery-<NG> component as NodeFeature resources.
3. **node-feature-discovery-gc** (Deployment): A component that consists of a single **gc** container, which deletes obsolete NodeFeature resources when a node is deleted.
4. **gpu-feature-discovery-<NG>** (DaemonSet): A component that queries the GPU driver via the NVIDIA Management Library (NVML) and writes information about GPU hardware capabilities to the `/etc/kubernetes/node-feature-discovery/features.d/gfd` file. Node-feature-discovery-worker publishes this information as NodeFeature resources, from which node-feature-discovery-master updates the corresponding `nvidia.com/*` labels for cluster nodes.

   The component is created by the Deckhouse controller of the `deckhouse` module for every NodeGroup (NG) whose configuration specifies the `.spec.gpu` parameter.

   It consists of the following containers:

   - **gpu-feature-discovery-init**: Init container that prepares the configuration for the main gpu-feature-discovery-ctr container.
   - **gpu-feature-discovery-ctr**: Main container.
   - **gpu-feature-discovery-sidecar**: Sidecar container that watches configuration changes and restarts the main container to apply them.
5. **nvidia-device-plugin-<NG>** (DaemonSet): A component that registers with kubelet through the Kubernetes Device Plugin API and publishes GPU resources for kube-scheduler.

   To work with resources, kubelet calls the `ListAndWatch`, `Allocate`, and `GetPreferredAllocation` gRPC methods on nvidia-device-plugin-ctr. After that, the component updates the number of available resources on the node and returns this information through kubelet.

   It consists of the following containers:

   - **nvidia-device-plugin-init**: Init container that prepares the configuration for the main nvidia-device-plugin-ctr container.
   - **nvidia-device-plugin-ctr**: Main container.
   - **nvidia-device-plugin-sidecar**: Sidecar container that watches configuration changes and restarts the main container to apply them.
6. **nvidia-mig-manager** (DaemonSet): An optional component that manages changes to the Multi-Instance GPU (MIG) profile on nodes with GPUs A100 and H100.

   The component performs the following actions:

   - Retrieves the desired MIG profile (the `nvidia.com/mig.config` label) and the current state.
   - Puts the node into maintenance mode when necessary (applies a taint, cordon, and drain).
   - Stops pods that use GPUs on the node.
   - Applies the MIG profile.
   - Initiates a node reboot when necessary.
   - Returns the node to service.

   It consists of the following containers:

   - **nvidia-mig-manager-init**: Init container that prepares executables and libraries.
   - **nvidia-mig-manager**: Main container.
7. **nvidia-dcgm** (DaemonSet): A component that consists of a single **nvidia-dcgm** container, which runs Data Center GPU Manager (DCGM). DCGM collects GPU health and utilization data (Error Correction Code (ECC), power).

8. **nvidia-dcgm-exporter** (DaemonSet): A component that consists of a single **exporter** container, which retrieves GPU metrics from the nvidia-dcgm component, and exposes them in Prometheus format.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Authorizes requests to retrieve metrics.
   - Works with NodeFeature and NodeFeatureRule resources.
   - Watches and updates Node resources.
   - Terminates pods that use GPU resources when the MIG profile changes.
2. **Kubelet**: Registers the module through the Device Plugin API.

The following external components interact with the module:

1. **Kubelet**: Calls the `ListAndWatch`, `Allocate`, and `GetPreferredAllocation` gRPC methods.
2. **Prometheus-main**: Collects metrics from nvidia-dcgm.

Additional resources

- Module `deckhouse` documentation
- Module `gpu` documentation
- Module `node-manager` custom resources

Terraform-manager module

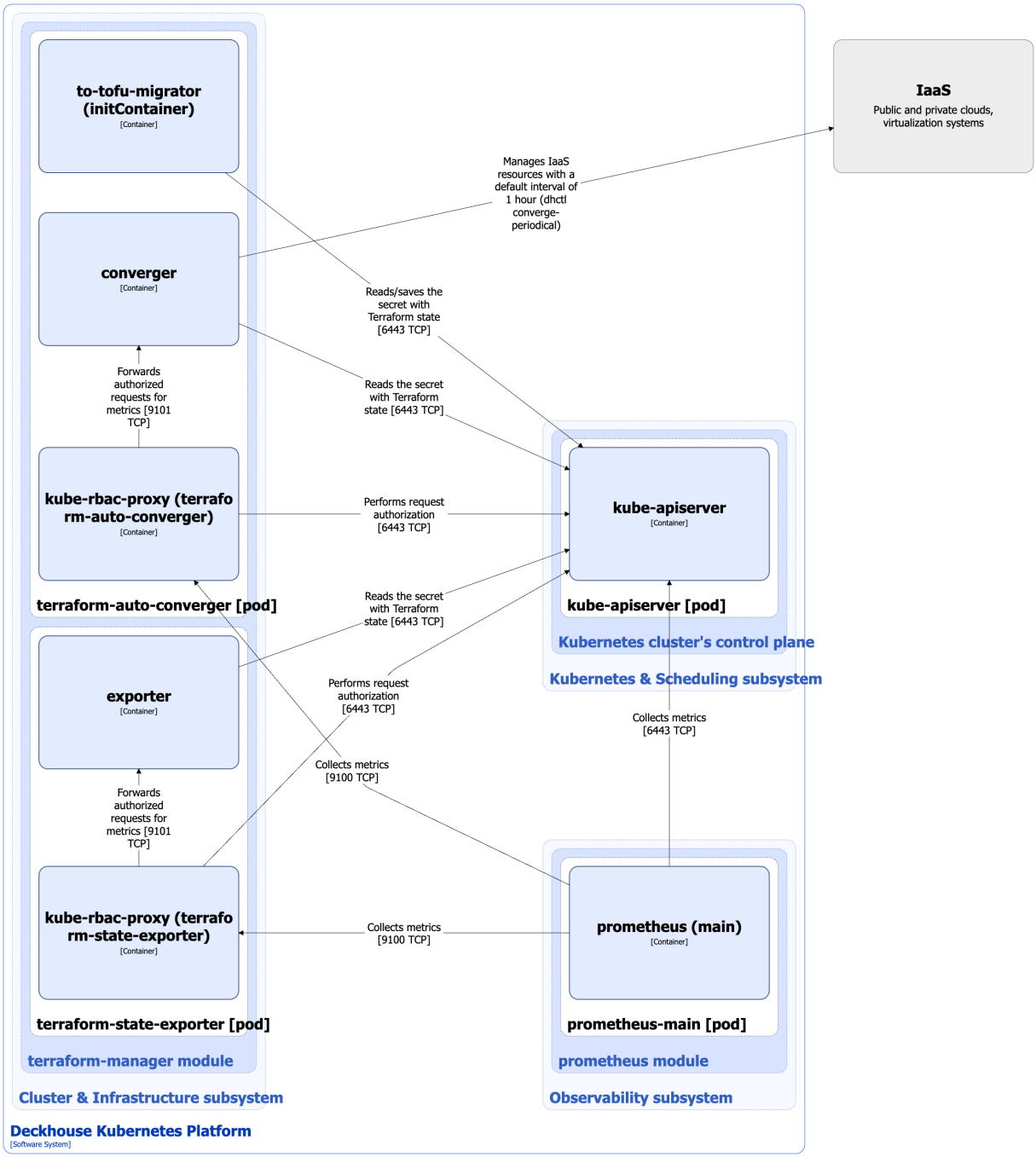
The `terraform-manager` module provides tools for managing the Terraform state in a DKP cluster.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `terraform-manager` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Terraform-auto-converger**: Periodically (once per hour by default) checks the Terraform state and applies non-destructive changes to infrastructure resources.

   The component operates only on the base infrastructure of the cluster. Cluster nodes are not automatically reconciled to the desired state. The check interval is configured using the `autoConvergerPeriod` parameter.

   It consists of the following containers:

   - **to-tofu-migrator**: Init container used to migrate the Terraform state to OpenTofu. It runs the `dhctl` utility with the `converge-migration` command.
   - **converger**: Main container running the `dhctl` utility with the `converge-periodical` command.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the metrics of the converger container. This component is an open-source project.

2. **Terraform-state-exporter**: Checks the Terraform state and exports related metrics.

   It consists of the following containers:

   - **exporter**: Main container running the `dhctl` utility with the `terraform converge-exporter` command.

    - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the metrics of the exporter container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
    - Reading and writing the Secret containing the Terraform state.
    - Authorization of metric requests.
2. **Cloud infrastructure** (or virtualization platform): Manages base infrastructure resources and reconciles them to the desired state.

The following external components interact with the module:

- **prometheus-main**: Collects metrics from terraform-auto-converger and terraform-state-exporter.

Additional resources

- Module `terraform-manager` documentation

IAM subsystem

This subsection describes the architecture of the IAM subsystem (Identity and Access Management) of Deckhouse Kubernetes Platform (DKP).

The IAM subsystem provides the following features in DKP:

- User authentication
- Role-based access control (RBAC)
- Multitenancy
- Automatic assignment of annotations and labels to namespaces

The IAM subsystem includes the following modules that implement the features described above:

- `user-authn`: User authentication.
- `user-authz`: Role-based access control (RBAC).
- `multitenancy-manager`: Multitenancy.
- `namespace-configurator`: Automatic assignment of annotations and labels to namespaces.

Authentication

Connecting to Kubernetes API using a generated kubeconfig

1. **Initialization**. Before the kube-apiserver starts, it requests the configuration endpoint of the OIDC provider (in this case — Dex) to retrieve the `issuer` and JWKS endpoint settings for token validation.
2. **Kubeconfig generation**. The Deckhouse Kubernetes Platform (DKP) web UI generates a kubeconfig file that includes an `ID token` and a `refresh token`. This file is used by `kubectl` or other Kubernetes clients.
3. **Authentication when accessing the API**. Upon receiving a request with an `ID token`, the `kube-apiserver` verifies the token's signature using keys from the JWKS endpoint. It then compares the `iss` (issuer) and `aud` (audience) claims in the token against the server configuration.

Protection against credential brute-forcing in Dex

Each user is allowed a maximum of 20 login attempts. Once this limit is reached, an additional attempt is allowed every 6 seconds.

Authentication via DexAuthenticator

1. **Login process via Dex**. In most cases, Dex redirects the user to the login page of an external identity provider (e.g., GitHub, Okta, Keycloak), then expects the user to return to the `/callback` URL after successful authentication.
   However, for providers like LDAP or Atlassian Crowd, this flow is not supported. Instead, the user enters their login and password in the Dex login form, and Dex performs authentication by calling the provider's API directly.
2. **Token and session storage**. DexAuthenticator sets a cookie containing the full `refresh token`, rather than issuing a short-lived ticket as with the `ID token`.
   This is because Redis, used by DexAuthenticator, does not persist data to disk.

If the `ID token` is missing from Redis, the user can obtain a new one using the `refresh token` stored in the cookie.

3. **Passing the token to the application**. DexAuthenticator sets the `Authorization` HTTP header with the `ID token` from Redis.
   This may be optional for some applications, such as Upmeter, where other authorization mechanisms are used.
   However, for applications like the Kubernetes Dashboard, this behavior is critical, as the Dashboard forwards the `ID token` to access the Kubernetes API on behalf of the user.

DKP-specific Dex extensions

DKP uses a modified version of Dex that supports:

- Groups for static user accounts and the Bitbucket Cloud provider (via the `bitbucketCloud` parameter).
- Passing the `group` claim to clients.
- The `obsolete tokens` mechanism, which helps prevent race conditions when refreshing tokens in OIDC clients.

High availability mode

DKP supports a high-availability mode via the `highAvailability` setting.
When enabled, multiple authenticator instances are deployed with redundancy to ensure continuous service.
If any of the authenticators fail, active user authentication sessions are preserved and remain uninterrupted.

Additional resources

- Module `user-authn` custom resources

Multitenancy

The `multitenancy-manager` module allows you to create isolated projects within the Deckhouse Kubernetes Platform (DKP). Projects provide resource quotas, network isolation, and security features that go beyond standard namespaces.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Internal logic

Project creation

The following custom resources are used to create a project:

- ProjectTemplate: Describes the project template. It defines the list of resources to be created in the project and the parameter schema that can be passed during project creation.
- Project: Describes a specific project.

When a Project is created from a specified ProjectTemplate, the following occurs:

1. The provided parameters are validated against the OpenAPI specification (defined in the `openAPIV3Schema` parameter of the ProjectTemplate resource).
2. The resource template is rendered using Helm. The rendering values are taken from the `parameters` field of the Project resource.
3. A namespace is created with the same name as the Project.
4. All resources described in the template are created one by one.

⚠ If you modify a project template, all previously created projects will be updated according to the new template.

Project isolation

Project isolation is based on the resource isolation mechanism provided by namespaces. Namespaces allow grouping pods, services, secrets, and other objects, but they do not offer full isolation. Projects extend namespace functionality by providing additional tools to improve control and security.

To manage project isolation scale, you can use the following Kubernetes features:

- **Access control resources** (AuthorizationRule / RoleBinding): Let you manage interactions between objects within a namespace. With them you can define rules and assign roles to precisely control who can do what in a project.
- **Usage control resources** (ResourceQuota): Let you define limits for CPU, RAM, and number of objects within a namespace. This helps prevent resource overconsumption and provides monitoring over project applications.
- **Network connectivity control resources** (NetworkPolicy): Let you manage incoming and outgoing

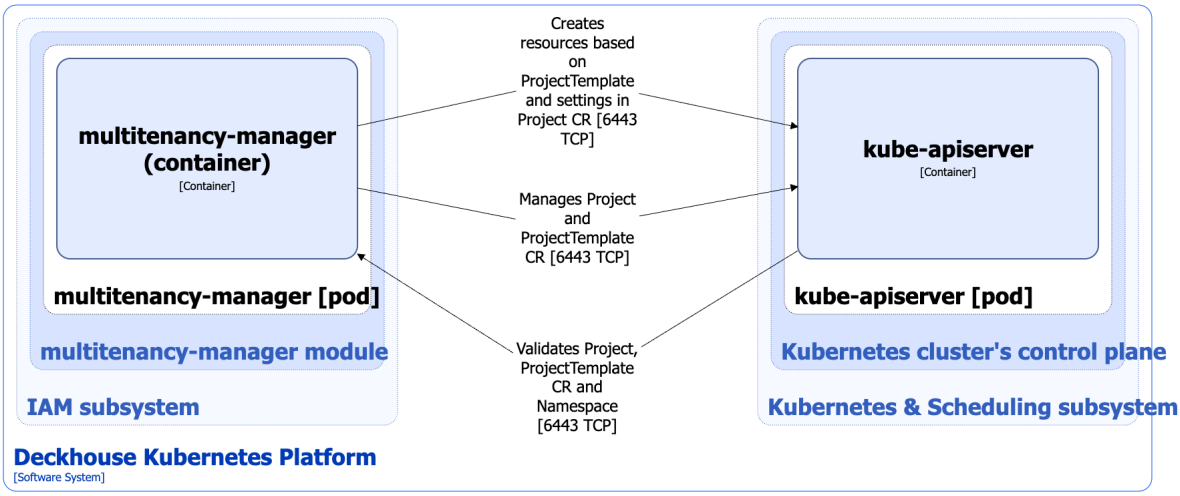
network traffic in a namespace. You can configure allowed connections between pods and improve security and network manageability within a project.

You can combine these tools to configure a project according to your application’s requirements.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The architecture of the `multitenancy-manager` module at Level 2 of the C4 model and its interactions with other DKP components are shown in the following diagram.



Module components

The module consists of the following components:

- **Multitenancy-manager**: The component consists of a single **multitenancy-manager** container and provides the following functions:
  - Managing the Project and ProjectTemplate custom resources.
  - Validating the Project and ProjectTemplate custom resources.
  - Validating Namespace if `.spec.settings.allowNamespacesWithoutProjects=false` is set in the `multitenancy-manager` module parameters.
  - Creating the resources specified in the ProjectTemplate custom resource based on the parameters set in Project.

> **Warning.** Multitenancy-manager has `cluster-admin` permissions, which allow it to create any objects described in the ProjectTemplate resource.

Module interactions

The module interacts with the following components:

- **Kube-apiserver**:
  - Managing the Project and ProjectTemplate custom resources.
  - Validating the Project and ProjectTemplate custom resources and the standard Namespace resource.
  - Creating the resources specified in the ProjectTemplate custom resource based on the parameters set in Project.

Additional resources

- Module `multitenancy-manager` documentation

User-authn module

The `user-authn` module implements a unified authentication system integrated with Kubernetes and the web interfaces used by modules of Deckhouse Kubernetes Platform (DKP), such as the `console` module.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

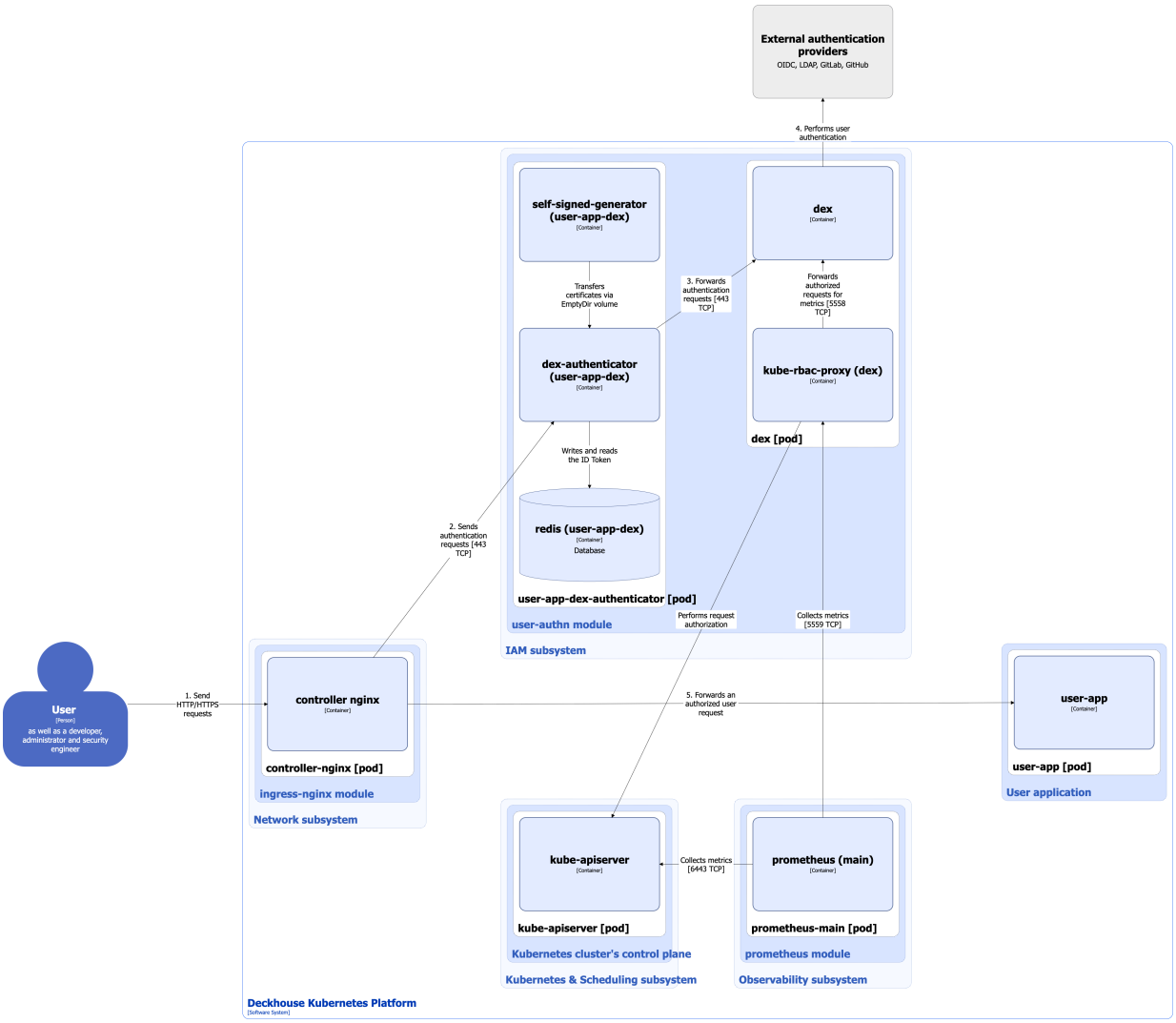
> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

In DKP, two authentication schemes are used for platform services and user applications:

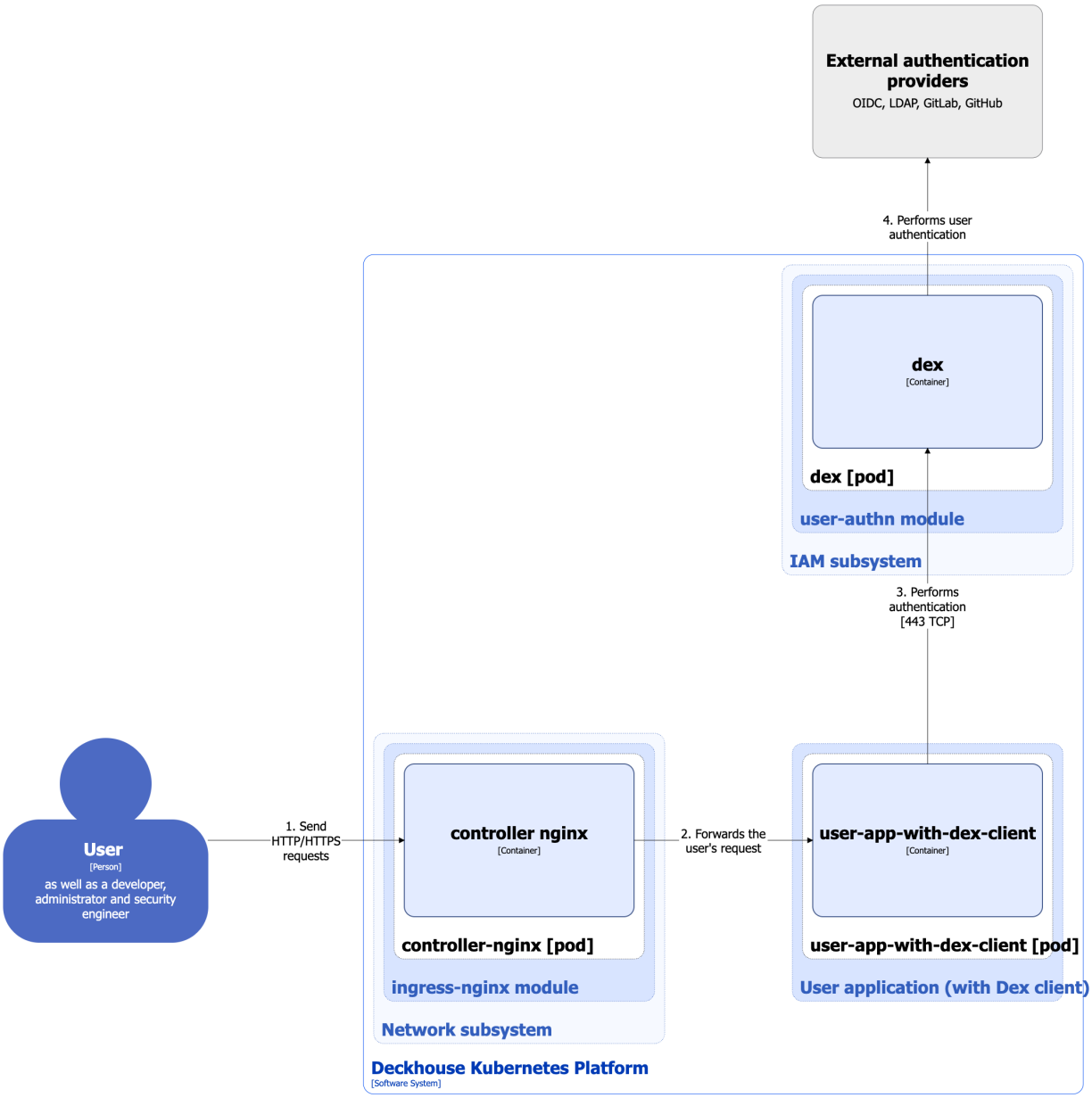
- Using dex-authenticator
- Using the Dex client

The architecture of the `user-authn` module at Level 2 of the C4 model and its interactions with other DKP components are shown in the following diagrams.

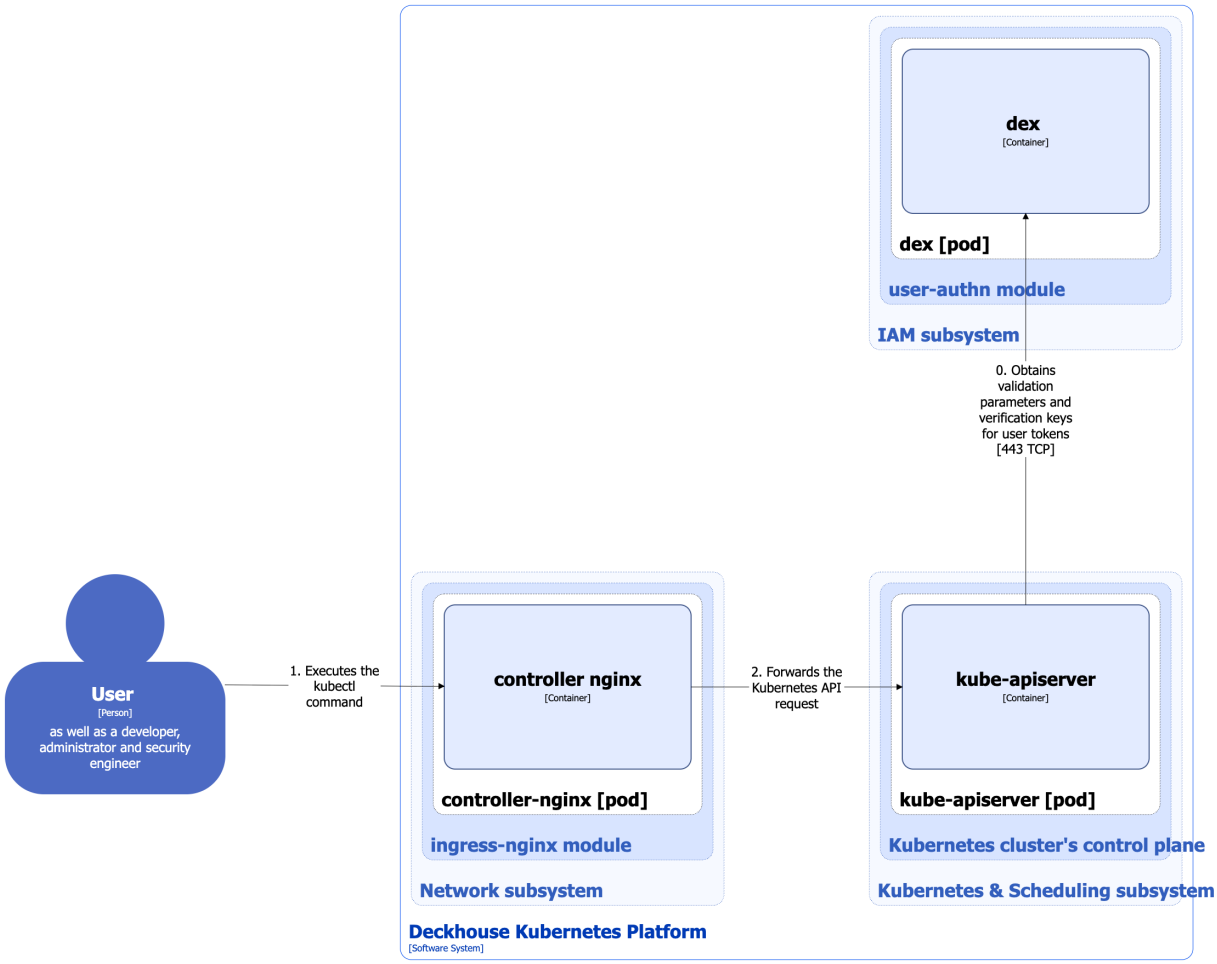
Using dex-authenticator:



Using the Dex client (for simplicity, the interaction between the `prometheus-main` module and dex is not shown in the diagram):



When connecting to the Kubernetes API using `kubectl` or other Kubernetes clients with a generated kubeconfig, a separate authentication scheme is used. It is described in detail in the corresponding documentation section:



Module components

The module consists of the following components:

1. **Dex**: Federated OpenID Connect provider that supports static users and integration with external authentication providers such as SAML, GitLab, or GitHub.

   It consists of the following containers:

   - **dex**: Main container implementing Dex functions.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to provider metrics.

2. **Dex-authenticator**: Middleware service used to authenticate requests to applications through the DKP cluster authentication service.

   When the Ingress controller is configured accordingly (using the NGINX `auth_request` module), requests are first forwarded to dex-authenticator for authentication.

   It consists of the following containers:

   - **dex-authenticator**: Main container of the service.
   - **redis**: Sidecar container with a Redis database used for temporary storage of ID tokens and fast access to them (since the database resides in memory).
   - **self-signed-generator**: Init container that generates a self-signed certificate when the pod starts.

Module interactions

The module interacts with the following components:

1. External authentication providers.
2. **Kube-apiserver**: Authorization of requests to retrieve metrics.

The following external components interact with the module:

1. **Ingress controller**: Forwards authentication requests to dex-authenticator for DKP platform services and user applications.

2. **User applications**: Can authenticate directly with dex (without dex-authenticator) if an OAuth2 client is configured in Dex for the application. For more details about configuring a Dex client, refer to the `user-`

`authn` module documentation.

3. **Kube-apiserver**: Queries dex when processing Kubernetes API requests made using a kubeconfig file:
   - When starting, kube-apiserver requests the OIDC provider configuration endpoint (Dex in this case) to obtain the `issuer` and the parameters required to validate tokens via the JWKS endpoint.
   - When receiving a request with an ID token, kube-apiserver verifies its signature using the keys obtained from the JWKS endpoint.

   More details about connecting to the Kubernetes API using a generated kubeconfig can be found in the corresponding documentation page.
4. **Prometheus-main**: Collects metrics from the dex provider.

Additional resources

- Module `console` documentation
- Module `user-authn` documentation

Security subsystem

This subsection describes the architecture of the Security subsystem of Deckhouse Kubernetes Platform (DKP).

The Security subsystem includes the following modules:

- `admission-policy-engine`: Lets you use security policies in the cluster according to Kubernetes Pod Security Standards. The module uses Gatekeeper to enforce these policies.
- `runtime-audit-engine`: Implements runtime audit (threat detection based on Linux kernel events and Kubernetes API audit events using Falco). The architecture is described in “Runtime audit”.
- `security-events-manager`: Performs declarative collection, processing, normalization, and delivery of security events from logs of applications and Kubernetes infrastructure components. The architecture is described in “Security events”.
- `operator-trivy`: Performs periodic vulnerability scanning of the DKP cluster.
- `cert-manager`: Manages TLS certificates in the cluster.
- `secrets-store-integration`: Delivers secrets to Kubernetes applications by integrating secrets, keys, and certificates stored in external secret stores.
- `secret-copier`: Automatically copies secrets to cluster namespaces.

Integrity control

Integrity control is a set of mechanisms for verifying containers to ensure their security and compliance with the specified configuration.

Deckhouse Kubernetes Platform (DKP) provides the following mechanisms:

- Integrity control of user workloads at startup: image signature verification during Kubernetes API request validation.
- Integrity control of running containers: runtime audit.
- Integrity protection of DKP modules: modules are installed as immutable images.

Integrity control of user workloads at startup

Image signature verification at the CRI level is not implemented in DKP.

The integrity and authenticity of user application images are verified during Kubernetes API request validation, that is, before a Pod is created, rather than at the CRI level. The mechanism is implemented by the `admission-policy-engine` module and relies on signatures created with Cosign.

The sequence of integrity checks at startup:

1. At the build stage, the image is signed with a private key using Cosign. The signature is published in the container image registry as a separate tag and does not modify the image itself.
2. A SecurityPolicy resource with the `policies.verifyImageSignatures` parameter is created in the cluster, listing the public keys and the image address patterns.
3. When a Pod is created or updated, the validating webhook passes the request to Gatekeeper, which in turn queries the Ratify component.
4. Ratify verifies the image signature against the specified public keys.
5. If the signature is missing or does not match the specified keys, Pod creation is denied.

Additionally, you can restrict the list of container image registries that Pods are allowed to be started from using the `policies.allowedRepos` parameter of the OperationPolicy resource.

For more details on the configuration, refer to “Image signature verification”.

At the node level, DKP also protects the integrity of unpacked image layers:

- When using containerd v2, the EROFS snapshot is used — each layer of the OCI image is mounted read-only. This reduces the risk of tampering with images already unpacked on a node, but does not eliminate it completely and does not replace image authenticity verification. For details on switching to containerd

v2, refer to “Migrating container runtime to containerd v2”.
- The SHA-256 hash verification performed by containerd when pulling an image protects against data corruption in transit, but does not confirm the image authenticity: it does not make it possible to determine who built the image.

These measures complement signature verification, but do not replace it. To verify the authenticity of images, be sure to use the API validation described above.

Integrity control of running containers

Runtime auditing in DKP includes analyzing Linux kernel events and Kubernetes API audit events. This makes it possible to track whether applications in pods are running unchanged, conform to their expected state, and have not been modified.

Auditing uses:

- Built-in rules
- Custom rules that can be added using the Falco condition syntax

Integrity control of running containers can detect threats such as launching command-line shells inside containers or pods, discovering containers running in privileged mode, mounting insecure paths into containers, or attempts to read sensitive data.

For more details on configuring runtime audit, refer to “Runtime audit”.

Integrity protection of DKP modules

Starting with version 1.74, DKP modules are installed not as a directory with files, but as an image in the EROFS format that is mounted read-only through the DM-Verity kernel mechanism. This protects the contents of an already installed module from being modified on the node: the module files cannot be overwritten in the mounted image, and DM-Verity verifies the image blocks when they are accessed.

The mechanism is enabled automatically if the `erofs` filesystem is registered in the kernel on the node running the DKP controller (a master node by default), that is, the corresponding kernel module is either loaded or built into the kernel. To check it, run the following command:

```
grep -w erofs /proc/filesystems
```

If the `erofs` filesystem is not available, DKP will continue to operate and will install modules the regular way, without protecting their integrity. The switch happens without a dedicated alert, so make sure to check `erofs` availability in advance.

> ⚠ DKP loads the `erofs` kernel module and configures it to load automatically only on nodes that use containerd v2 as the container runtime. If master nodes use containerd v1, the `erofs` module may end up not loaded, and DKP modules will be installed without integrity protection, even if the kernel supports `erofs`.
>
> To make sure the mechanism is engaged, either switch master nodes to containerd v2, or have the operating system load the `erofs` module.

Note the following when using this mechanism:

- The mechanism protects modules from being modified after installation, but does not confirm their authenticity: there is no cryptographic signature verification for modules. The container image registry the module is pulled from remains the source of trust.
- The mechanism applies only to DKP modules and does not cover user application images.
- The mechanism is available in all DKP editions and does not require to be enabled separately.

Additional resources

- Module `admission-policy-engine` custom resources

Runtime audit architecture

The runtime audit of Deckhouse Kubernetes Platform (DKP) is based on the Falco threat detection system. This mechanism analyzes Linux kernel events and Kubernetes API audit events to detect suspicious activity in running containers and across the cluster.

DKP deploys Falco agents on each node as part of a DaemonSet. Once started, the agents begin collecting OS system calls and Kubernetes audit data.

> **i** Falco developers recommend running it as a systemd service, which can be challenging in Kubernetes clusters that support autoscaling. DKP includes additional security mechanisms such as multitenancy and resource control policies. Combined with the DaemonSet deployment, these mechanisms ensure a high level of protection.

Each cluster node runs a Falco Pod with the following components:

- `falco`: Collects events, enriches them with metadata, and outputs them to stdout.
- `rules-loader`: Retrieves rule data from FalcoAuditRules custom resources and stores them in a shared directory.
- `falcosidekick`: Receives events from `falco` and exports them as metrics to external systems.
- `kube-rbac-proxy`: Protects the `falcosidekick` metrics endpoint from unauthorized access.

For a detailed architecture of the `runtime-audit-engine` module, which implements DKP security event audit, refer to the corresponding documentation section.

Audit rules

Runtime audit event analysis is performed using rules that define suspicious behavior patterns. Each rule consists of a condition expression written in accordance with Falco's condition syntax.

Built-in rules

DKP provides the following types of built-in rules:

- **Kubernetes audit rules**: Help detect security issues in DKP and in the audit mechanism itself. These rules are located in the `falco` container at `/etc/falco/k8s_audit_rules.yaml`.

Custom rules

Custom rules can be defined using the FalcoAuditRules custom resource.

Each Falco agent includes a sidecar container with a `shell-operator` instance. This instance reads rules from Kubernetes resources, converts them into Falco rule format, and stores them in the `/etc/falco/rules.d/` directory inside the Pod. When a new rule is added, Falco automatically reloads the configuration.

Additional resources

- Module `runtime-audit-engine` documentation

Security events architecture

The `security-events-manager` module within Deckhouse Kubernetes Platform (DKP) performs declarative collection, processing, normalization, and delivery of security events extracted from logs of applications and Kubernetes infrastructure components.

A security event is a structured record of an action or fact that is significant from an information security perspective. Typical categories of such events include:

- Authentication and authorization
- Access to APIs and configuration
- Changes to cluster objects
- Runtime and network activity anomalies

Regardless of the original log format, the output is a uniform event model with a mandatory minimum set of attributes.

> **⚠** The `security-events-manager` module is in the `Experimental` stage and may change in future releases.

Solution architecture

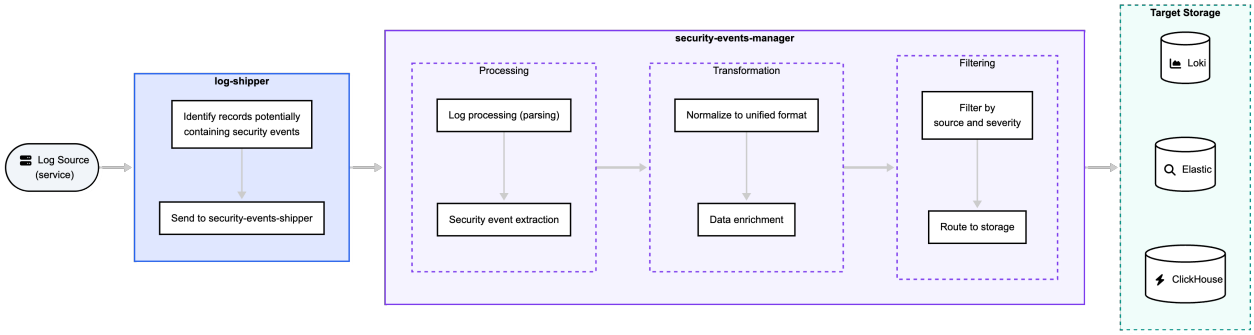
The architecture is intended to build a unified pipeline for working with security events extracted from the logs of applications and Kubernetes infrastructure components. The module performs three processing stages:

1. **Collection**: Collecting logs from pod sources and node files, initial selection of records that may contain security events.
2. **Processing and enrichment**: Parsing logs, extracting event signals, transforming into a unified model, and enriching with context.

3. **Delivery**: Policy-based filtering and delivery to configured storages.

At the delivery stage, sending to several types of storage and analytics systems is supported (such as Loki, Elasticsearch, Kafka, Splunk, Vector, File).

Processing pipeline



The architecture separates three stages of the pipeline: **collection**, **processing and enrichment**, and **delivery**.

Collection layer (log-shipper)

Collection is performed via the auxiliary `log-shipper` module:

- Container sources use application logs within the namespace.
- Cluster sources use node files and system service logs (for example, `/var/log/kube-audit/audit.log`, `/var/log/auth.log`).

A two-tier scheme is used:

- The `log-shipper` module performs preliminary selection of log records using simple comparison operators (`In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`) and forwards them to the gateway.
- The `security-events-manager` module (gateway) performs field recognition (parsing), further processing, and delivery.

Because log parsing is resource-intensive, it is performed only for records pre-selected as potentially containing security events, not for all incoming logs. Therefore, the initial selection stage does not perform deep field-value filtering that requires full content parsing.

Example of an input log for the selection stage:

```
{
  "time": "2026-05-10T14:21:03Z",
  "kind": "Event",
  "source": "kube-apiserver",
  "level": "Metadata",
  "message": "Unauthorized",
  "reason": "Unauthorized",
  "code": 401,
  "requestURI": "/api/v1/namespaces/default/secrets",
  "user": "system:serviceaccount:default:demo",
  "sourceIPs": [
    "206.123.145.70"
  ]
}
```

Processing and extraction

After being forwarded to the gateway, the log structure is recognized. Standard processing strategies are supported:

- `JSON`: For structured logs.
- `Regex`: For string formats with predictable patterns.
- `Grok`: For complex, non-unified formats.

The processing result is used to derive event signals and build a unified set of fields.

Example output of the processing stage:

```
{
  "parsed": {
    "timestamp": "2026-05-10T14:21:03Z",
    "source_component": "kube-apiserver",
    "http_status": 401,
    "request_uri": "/api/v1/namespaces/default/secrets",
    "actor_id": "system:serviceaccount:default:demo",
    "source_ip": "206.123.145.70"
  }
}
```

After processing, the record becomes a source of fields for event classification (code/category/severity/outcome), as well as for building context (`actor`, network attributes, and source metadata).

Transformation and enrichment

Transformation is implemented in two steps:

1. **Transform**: Mapping fields from the original log to the fields of the target event model.
2. **Enrich**: Adding or refining fields from additional context sources (for example, static environment attributes, subject roles, and service indicators).

The order is fixed: `Transform` is applied first, then `Enrich`. In case of a conflict for a target field, the final value is determined by the `Enrich` stage.

Filtering and delivery

After the event is formed, a delivery policy is applied:

- Source filtering
- Filtering by minimum severity
- Routing to one or more destinations

Filtering rules can use both exact source identifiers and source masks, which makes it possible to manage flows at the level of individual services or entire groups.

Destinations can be storage and processing systems such as cluster Loki, external SIEM/log platforms, and streaming buses. The delivery scheme supports parallel delivery to multiple target systems.

Security event model

Below is the structure of the event that is sent to storage.

Required fields:

- `id`, `timestamp`
- `source.component`
- `event.code`, `event.category`, `event.severity`, `event.outcome`
- `eventMetadata.cluster`

Optional fields:

- `eventMetadata.sourceIPs`
- `actor.*`
- `object.*`

```
{
  "id": "2f0de5c2-2e58-4d3f-b4fe-5ec6f1935b9f",
  "timestamp": "2026-05-10T14:21:03Z",
  "source": {
    "component": "kube-apiserver"
  },
  "event": {
    "code": "UNAUTHORIZED_ACCESS",
    "category": "Rbac",
    "severity": "High",
    "outcome": "Failure"
  },
  "eventMetadata": {
    "cluster": "prod-cluster",
    "sourceIPs": [
      "206.123.145.70"
    ]
  },
  "actor": {
    "id": "system:serviceaccount:default:demo",
    "type": "ServiceAccount"
  },
  "object": {
    "id": "/api/v1/namespaces/default/secrets",
    "type": "KubernetesResource"
  }
}
```


Minimal sufficient lifecycle

In a practical scenario, the architecture works along the following chain:

1. Log sources are connected to the collection pipeline.
2. An initial selection of potentially relevant records is performed.
3. Records are processed and transformed into unified events.
4. Events are enriched with contextual attributes.
5. Filtering and routing rules are applied.
6. Events are delivered to target storages and analytics systems.

The result is a unified and manageable flow of security events suitable for monitoring, investigations, and long-term auditing.

Supported security events

The `security-events-manager` module ships with a built-in set of security event detection rules covering authentication, configuration, RBAC, runtime, and other categories. For the up-to-date list of implemented security events, their codes, severity levels, and descriptions, refer to the module documentation.

Additional resources

- Module `log-shipper` documentation
- Module `security-events-manager` documentation

Admission-policy-engine module

The `admission-policy-engine` module enforces security policies and operational restrictions in a Kubernetes cluster. Policies are applied based on Pod Security Standards checks and rules from the following custom resources:

- OperationPolicy: Describes the operational policy of the cluster.
- SecurityPolicy: Describes the security policy of the cluster.
- SecurityPolicyException: Describes exceptions to the cluster security policy.

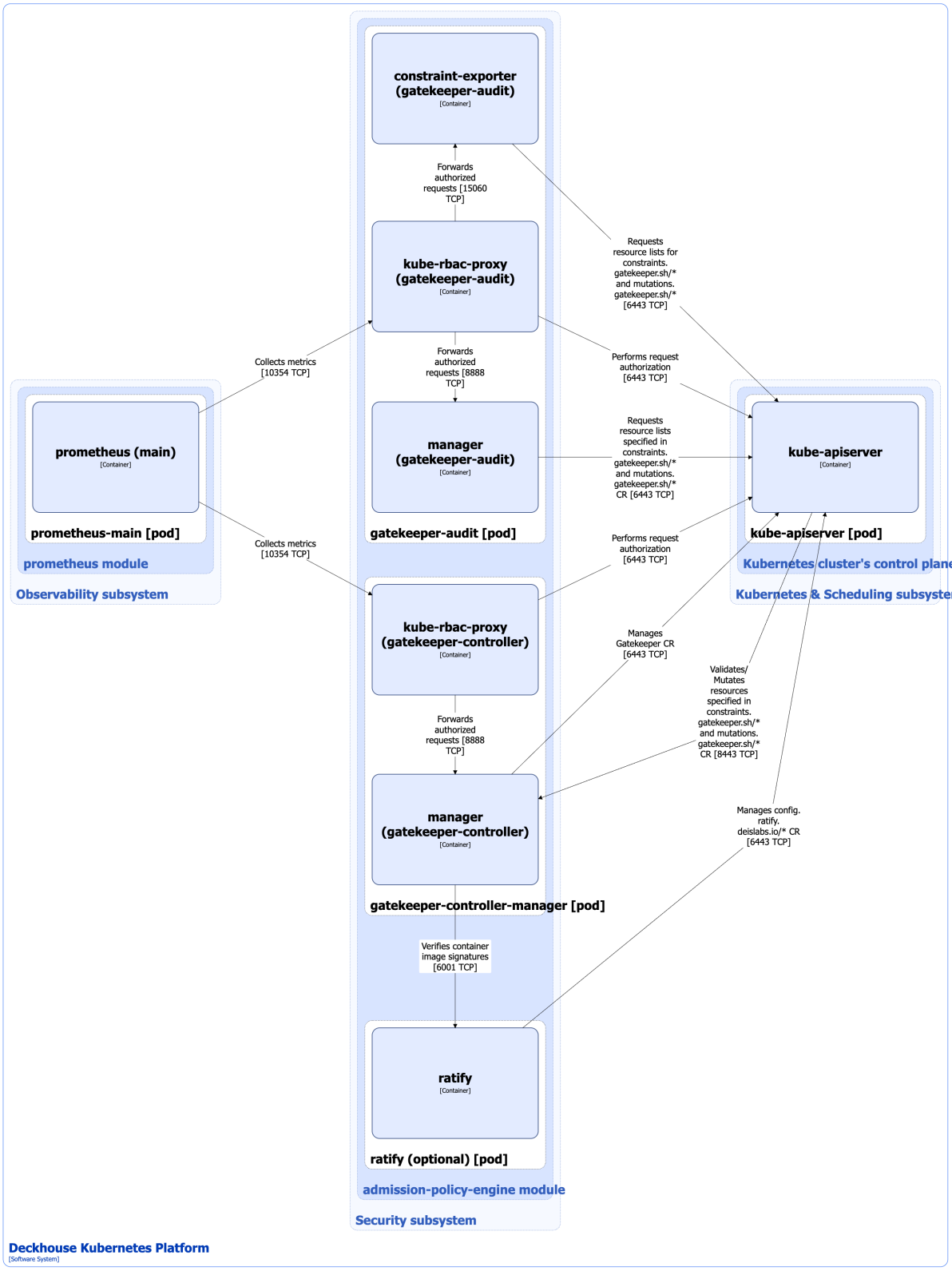
> **i**
>
> The `deckhouse` module processes the OperationPolicy and SecurityPolicy resources. The Deckhouse controller of the `deckhouse` module uses addon-operator and module hooks to create custom resources for Gatekeeper based on OperationPolicy and SecurityPolicy. Gatekeeper uses the resulting custom resources to validate newly created or updated Kubernetes resources.
>
> For details on the hooks concept, refer to the addon-operator documentation.

For a detailed description of the module, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `admission-policy-engine` module and its interaction with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Gatekeeper-controller-manager**: A Gatekeeper controller that performs the following operations:

   - Manages Gatekeeper custom resources.
   - Validates Kubernetes resources specified in custom resources from the `constraints.gatekeeper.sh/*` API group.
   - Mutates Kubernetes resources specified in the AssignMetadata, Assign, ModifySet, and AssignImage custom resources.

   Security rules are defined using the ConstraintTemplate custom resource and custom resources from the `constraints.gatekeeper.sh/*` API group. A ConstraintTemplate defines new policy types, based on which specific security policies are created to validate resources.

   It consists of the following containers:

   - **manager**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to controller metrics.

2. **Gatekeeper-audit**: Implements periodic checks of existing Kubernetes resources for compliance with security policies.

   It consists of the following containers:

   - **manager**: Main container.
   - **constraint-exporter**: Sidecar container that exposes additional metrics for the `constraints.gatekeeper.sh/*` and `mutations.gatekeeper.sh/*` custom resources.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to metrics from `manager` and `constraint-exporter`.

3. **Ratify**: An optional component consisting of a single **ratify** container. It provides a Gatekeeper provider implementation for validating metadata of used artifacts. In DKP, this provider is used to verify container image signatures and is available in the SE+, EE, CSE Lite, and CSE Pro editions.

> **i** Gatekeeper uses the Provider custom resource to extend resource validation capabilities in Kubernetes. The Provider resource describes the service endpoint to which Gatekeeper sends requests during ValidationWebhook execution. Some DKP modules, such as `operator-trivy`, can create Provider custom resources and thereby extend the verification capabilities.

Module interactions

The module interacts with the following components:

- **Kube-apiserver**:
  - Monitors Kubernetes resources specified in custom resources of the `constraints.gatekeeper.sh/*` and `mutations.gatekeeper.sh/*` API groups.
  - Manages ConstraintTemplate, Assign, AssignImage, AssignMetadata, ModifySet, as well as resources from the `constraints.gatekeeper.sh/*` and `config.ratify.deislabs.io/*` API groups.

The following external components interact with the module:

1. **Kube-apiserver**: Validates Kubernetes resources and checks their compliance with the defined security rules.
2. **Prometheus-main**: Collects module metrics.

Additional resources

- Module `admission-policy-engine` documentation
- Module `deckhouse` documentation

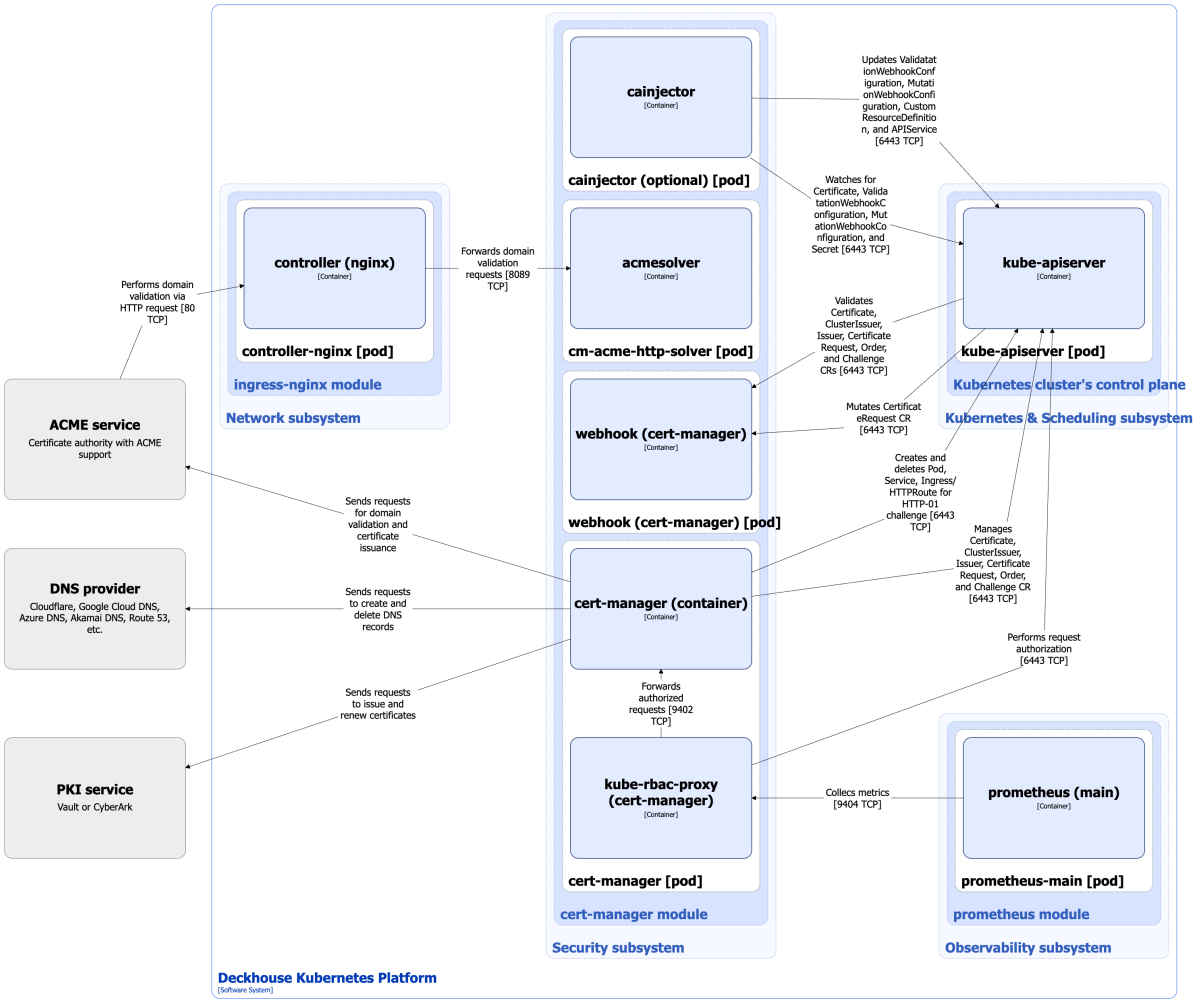
Cert-manager module

The `cert-manager` module automates the full certificate management lifecycle in a cluster: from issuing and renewing self-signed certificates to integration with external certificate authorities such as Let’s Encrypt, HashiCorp Vault, and Venafi. This significantly simplifies service security and provides centralized control over all certificate-related processes.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cert-manager` module and its interactions with other DKP components are shown in the following diagram:



Module components

The `cert-manager` module consists of the following components:

1. **Cert-manager**: A controller that provides the full certificate management lifecycle in Deckhouse Kubernetes Platform (DKP). `Cert-manager` manages the following custom resources:
   - Issuer: Describes settings and parameters for obtaining certificates from a specific source, such as a CA or an external service. It is used within a selected namespace.
   - ClusterIssuer: Similar to Issuer, but applies to the entire cluster and is available in all namespaces.
   - Certificate: Defines the required certificate and specifies parameters such as the subject, validity period, the Issuer or ClusterIssuer used, and additional options.
   - CertificateRequest: A request to issue or renew a certificate.
   - Challenge: Describes a task used to complete domain validation, such as an HTTP-01 or DNS-01 challenge for Let’s Encrypt.
   - Order: Groups related Challenges into a sequence for obtaining a certificate from an ACME server, such as Let’s Encrypt.

   The component includes the following containers:
   - **cert-manager**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the cert-manager container metrics.

> **i** To complete the DNS-01 challenge, `cert-manager` supports a number of popular DNS providers, such as AzureDNS, Cloudflare, DigitalOcean, and others. For the full list of supported DNS providers, refer to the cert-manager documentation. For providers that are not supported out of the box, webhook issuer types are used as external components. To work correctly, such providers must be installed in the `d8-cert-manager` system namespace. When making changes to this namespace, it is recommended to always account for such extensions so they remain operational.

2. **Webhook**: A component consisting of a single webhook container and performing the following operations:
   - validates the Issuer, ClusterIssuer, Certificate, CertificateRequest, Challenge, and Order custom resources
   - mutates CertificateRequest custom resources by adding the user identity created the certificate request

   In DKP, validation is disabled for resources in the `d8-cert-manager` namespace and for namespaces with the `cert-manager.io/disable-validation=true` label.
3. **Cainjector**: An additional component consisting of a single cainjector container. Cainjector automatically injects or updates root certificate authority (CA) certificates in all relevant Kubernetes resources: ValidatingWebhookConfiguration, MutatingWebhookConfiguration, CustomResourceDefinition, and APIService. This keeps trusted root certificates up to date for services that use webhooks and API extensions.

   Cainjector is enabled using the `.spec.settings.enableCAInjector` parameter in the `cert-manager` module settings. DKP does not use cainjector, so enable it only if your services use custom CA injections.

   Cainjector processes only resources with the `cert-manager.io/inject-ca-from`, `cert-manager.io/inject-ca-from-secret`, or `cert-manager.io/inject-apiserver-ca` annotations, depending on the resource type.
4. **Cm-acme-http-solver**: A temporary pod with the acmesolver container that is launched to complete an HTTP-01 Challenge during domain validation through ACME, such as Let's Encrypt. This pod is automatically created by `cert-manager` controller for the duration of the HTTP-01 Challenge and is deleted after the procedure is complete. This approach provides secure temporary publication of the resource that confirms domain ownership for certificate issuance.

Module interactions

The `cert-manager` module interacts with the following components:

1. **Kube-apiserver**:
   - Manages the Issuer, ClusterIssuer, Certificate, CertificateRequest, Challenge, and Order custom resources.
   - Watches and updates ValidatingWebhookConfiguration, MutatingWebhookConfiguration, CustomResourceDefinition, and APIService resources.
2. **ACME service**: Handles requests for domain verification and certificate issuance.
3. **PKI service**: Handles requests for issuing and renewing certificates.
4. **DNS provider**: Handles requests to add and delete records in DNS services to complete DNS-01 Challenge during domain validation through ACME server.

The following external components interact with the module:

1. **Kube-apiserver**:
   - validates the Issuer, ClusterIssuer, Certificate, CertificateRequest, Challenge, and Order custom resources;
   - mutates CertificateRequest custom resources.
2. **Prometheus-main**: Collects `cert-manager` metrics.
3. **Nginx Controller**: Forwards requests from ACME server to `cm-acme-http-solver`.

Additional resources

- Module `cert-manager` documentation

Module operator-trivy

The `operator-trivy` module scans user images at runtime for known CVEs (Common Vulnerabilities and Exposures), including vulnerabilities specific to Astra Linux, ALT Linux, and RED OS. It is based on the Trivy project. Scanning uses public vulnerability databases, enriched with Astra Linux, ALT Linux, RED OS databases, and BDU FSTEC (Data Bank of Information Security Threats by the Federal Service for Technical and Export Control of Russia).

The module also analyzes Kubernetes cluster compliance with the CIS (Center for Internet Security) Kubernetes Benchmark requirements.

Additionally, the `operator-trivy` module can collect workload behavior models when the optional node-agent component is enabled.

The `operator-trivy` module works with custom resources from the `trivy.deckhouse.io` and `spdx.softwarecomposition.kubescape.io` API groups.

The `trivy.deckhouse.io` API group includes the following resources:

- ClusterComplianceReport: Stores an aggregated cluster compliance report for security requirements (for example, CIS Kubernetes Benchmark).
- ClusterConfigAuditReport: Stores cluster-level audit results for Kubernetes object configuration.
- ClusterInfraAssessmentReport: Stores cluster-level Kubernetes infrastructure security assessment results.
- ClusterRbacAssessmentReport: Stores cluster-level role-based access control (RBAC) settings assessment results.
- ClusterSbomReport: Stores an aggregated cluster-level SBOM (Software Bill of Materials) for software components.
- ClusterVulnerabilityReport: Stores aggregated cluster-level vulnerability scanning results.
- ConfigAuditReport: Stores audit results for Kubernetes object configuration.
- ExposedSecretReport: Stores results of searching for potential secrets in container images.
- InfraAssessmentReport: Stores security assessment results for Kubernetes infrastructure objects.
- NodeVulnerabilityReport: Stores vulnerability scanning results for node `rootfs` (the root filesystem).
- RbacAssessmentReport: Stores RBAC settings assessment results for excessive privileges and other risks.
- RegistryImageVulnerabilityReport: Stores CVE scanning results for a specific image tag in a registry.
- RegistryScanTarget: Defines a target registry, repositories, and periodic scanning parameters.
- SbomReport: Stores an SBOM, that is, the software and dependency composition of a container image.
- VulnerabilityReport: Stores vulnerability scanning results for container images.

The `spdx.softwarecomposition.kubescape.io` API group includes the following resources:

- ApplicationProfiles: Stores an application behavior profile at runtime (system calls, executed processes, file access, HTTP endpoints).
- CollapseConfigurations: Defines dynamic path, endpoint, and network address aggregation parameters to reduce runtime profile size.
- ConfigurationScanSummaries: Stores a summary of configuration scan results for a workload group within a specified scope (for example, a namespace).
- ContainerProfiles: Stores an individual container profile, including runtime behavior and network interactions.
- GeneratedNetworkPolicies: Stores NetworkPolicy objects generated from observed traffic.
- KnownServers: Stores a catalog of known servers and IP address ranges to enrich generated network policies.
- NetworkNeighborhoods: Stores an observed map of incoming and outgoing workload network interactions.
- OpenVulnerabilityExchangeContainers: Stores OpenVEX documents with vulnerability statuses for components.
- SbomSyftFiltereds: Stores a filtered Syft SBOM with only relevant vulnerable components.
- SbomSyfts: Stores SBOM in Syft format.
- SeccompProfiles: Stores container seccomp profiles (allowed system calls and filtering rules).
- VulnerabilityManifestSummaries: Stores VulnerabilityManifest summaries by severity level with links to related objects.
- VulnerabilityManifests: Stores a detailed manifest of discovered vulnerabilities.
- VulnerabilitySummaries: Stores an aggregated vulnerability summary for a specified scope.
- WorkloadConfigurationScans: Stores detailed configuration scan results for a specific workload.
- WorkloadConfigurationScanSummaries: Stores a summary of WorkloadConfigurationScan results.

For more details, see the module documentation section.

Module architecture

> **i** The following assumptions are made to simplify the diagram:
>
> - The diagram shows direct interaction between containers from different pods. In practice, they communicate through corresponding Kubernetes Services (internal load balancers). Service names are omitted when they are clear from context. In other cases, the service name is shown above the arrow.
> - Pods can run with multiple replicas, but all pods are shown as a single replica in the diagram.

The level 2 C4 architecture of the `operator-trivy` module and its interactions with other Deckhouse Kubernetes Platform (DKP) components are shown in the following diagram:

Module components

The module consists of the following components:

1. **Operator** (Deployment): A component based on Trivy Operator that watches Pod, ReplicaSet, ReplicationController, StatefulSet, DaemonSet, CronJob, and Job resources, and starts a Job task for scanning when their specifications change.

   Operator analyzes Job execution results and generates the following reports as custom resources:

   - ClusterComplianceReport.
   - ClusterConfigAuditReport.
   - ClusterInfraAssessmentReport.
   - ClusterRbacAssessmentReport.
   - ClusterVulnerabilityReport.
   - ConfigAuditReport.
   - ExposedSecretReport.
   - InfraAssessmentReport.
   - NodeVulnerabilityReport.
   - RbacAssessmentReport.
   - SbomReport.
   - VulnerabilityReport.

   Includes the following containers:

   - **operator**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to controller metrics.

2. **Trivy-server** (StatefulSet): This component implements a security scanning service and is based on the Open Source Trivy project.

   Trivy-server updates its vulnerability database from an Open Container Initiative (OCI) image at startup and then periodically.

   Trivy-server processes requests from other module components (operator, registry-scanner, trivy-provider, and scanning tasks), performs the requested scan, and returns the result.

   Includes the following containers:

   - **server**: Main container.
   - **trivy-db-info**: Sidecar container that synchronizes Trivy cache metadata with the `trivy-db-info` ConfigMap resource in the `d8-operator-trivy` namespace.

3. **Trivy-provider** (StatefulSet): This component includes a single **trivy-provider** container and provides an image verification interface for the Gatekeeper component of the `admission-policy-engine` module.

   When the module is installed, a Provider resource is created to register trivy-provider as a provider in Gatekeeper. For details about the integration, see the relevant Gatekeeper documentation section.

   The Deckhouse controller deploys this component if the `.settings.denyVulnerableImages.enabled` parameter of the ModuleConfig custom resource is set to `true` (default is `false`) and the Gatekeeper component of the `admission-policy-engine` module is enabled.

4. **Report-updater** (Deployment): An optional controller that includes a single **report-updater** container, implements a MutatingWebhook, and enriches the VulnerabilityReport custom resource with BDU identifiers. The BDU dictionary is regularly updated from an OCI image every six hours.

   The Deckhouse controller deploys this component if the `.settings.linkCVEtoBDU` parameter of the ModuleConfig custom resource is set to `true` (default is `false`).

5. **Registry-scanner** (StatefulSet): An optional component that includes a single **registry-scanner** container and regularly scans images located in arbitrary container registries specified by the user. Scanning of workloads in the cluster itself is performed by the operator component.

   The Deckhouse controller deploys this component if the `.settings.registryScanning.enabled` parameter of the ModuleConfig custom resource is set to `true` (default is `false`).

   Registry-scanner reads RegistryScanTarget custom resources, runs image scans through the trivy-server component, and stores processed results in the RegistryImageVulnerabilityReport custom resource.

6. **Security-storage** (Deployment): A controller that includes a single **apiserver** container, extends Kubernetes API, and processes requests for custom resources from the `trivy.deckhouse.io` and `spdx.softwarecomposition.kubescape.io` API groups.

   Security-storage also implements the backend for storing these resources:

   - Metadata is stored in an SQLite database.
   - Object body is stored in gob format in the `/data` directory.

7. **Node-agent** (DaemonSet): An optional component with a single **node-agent** container that runs on all cluster nodes in privileged mode, observes container behavior by using eBPF programs, and builds runtime profiles.

   Node-agent stores runtime profiles in ApplicationProfile and NetworkNeighborhood custom resources (in the security-storage component). For details about node-agent behavior, see the module documentation

section.

> ⚠ The node-agent component has privileged access to the operating system of each node. In Linux, this requires the following capabilities:
>
> - SYS_ADMIN
> - SYS_PTRACE
> - NET_ADMIN
> - SYSLOG
> - SYS_RESOURCE
> - IPC_LOCK
> - NET_RAW
>
> This is required to observe container behavior by using eBPF programs.
>
> Node-agent runs in profiling mode and does not perform active rule-based attack detection.

The Deckhouse controller deploys this component if the `.settings.nodeAgent.enabled` parameter of the ModuleConfig custom resource is set to `true` (default is `false`).

8. **Scan-noderootfs-<HASH>** (Job): A component that includes a single **node-rootfs-scanner** container and implements a task to scan the node root filesystem (scan-noderootfs). The task is created and managed by the operator component.
9. **Scan-vulnerabilityreport-<HASH>** (Job): A component that launches tasks for security scanning of Pod, ReplicaSet, ReplicationController, StatefulSet, DaemonSet, CronJob, and Job resources by using the trivy-server component. The task is created and managed by the `operator` component.

   Consists of a set of **<CONTAINER_NAME>** containers, each responsible for scanning the corresponding workload container. Their base image is Trivy with an added trivy-wrapper. The target container image from the workload specification is passed in arguments. Trivy-wrapper authenticates to the registry storage by using the `trivy registry login` command and then hands control over to Trivy.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Authorizes requests to operator metrics.
   - Works with custom resources from the `trivy.deckhouse.io` API group.
   - Works with RegistryScanTarget and RegistryImageVulnerabilityReport custom resources.
   - Watches Pod, ReplicaSet, ReplicationController, StatefulSet, DaemonSet, CronJob, and Job resources.
   - Creates, updates, and deletes Secret, ConfigMap, and Job resources.
2. **Registry**: Pulls OCI images with the vulnerability database and the BDU database.
3. **Container registry**: Scanning images.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Forwards API requests for custom resources from the `trivy.deckhouse.io` and `spdx.softwarecomposition.kubescape.io` API groups.
   - Mutates VulnerabilityReport custom resources.
2. **Prometheus-main**: Collects module metrics.

Additional resources

- Module `admission-policy-engine` documentation
- Module `operator-trivy` documentation
- Module `registry` documentation

Runtime-audit-engine module

The `runtime-audit-engine` module implements runtime auditing in Deckhouse Kubernetes Platform (DKP) based on the Falco threat detection system. The module collects Linux kernel events and Kubernetes API audit events (using the `k8saudit` plugin), enriches them with Kubernetes Pod metadata, and generates security events according to configured rules. Audit rules are defined using the FalcoAuditRules custom resource.

When the module is enabled, the `control-plane-configurator` ConfigMap is created in the `d8-runtime-audit-engine` namespace with the audit webhook URL and CA. The `control-plane-manager` module detects this ConfigMap and configures the control plane to send Kubernetes API audit events to the `runtime-audit-engine` module.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `runtime-audit-engine` module and its interactions with other DKP components are shown in the following diagram:

Module components

The `runtime-audit-engine` module consists of the following components:

1. **Runtime-audit-engine** (DaemonSet): A component deployed on each cluster node. It collects audit events, evaluates rules, outputs triggered rules to stdout, and exports them as Prometheus metrics. Data is received from the Linux kernel via interception of system calls (syscalls) and from `containerd` via a Unix socket.

   The component includes the following containers:

   - **falco**: Main container that collects security events from cluster nodes and containerized applications in DKP based on the Falco threat detection system.
   - **falcosidekick**: Sidecar container that receives events from the `falco` component and exports audit events as Prometheus metrics.
   - **rules-loader**: Sidecar container that performs the following operations:
     - Watches FalcoAuditRules custom resources and stores them in the shared `/etc/falco/rules.d/` Pod directory for processing by the `falco` component.
     - Validates the FalcoAuditRules custom resource.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC (Role-Based Access Control) that provides secure access to component metrics.

   > **⚠** The `falco` container has privileged access to each node operating system. The container security context includes the `BPF`, `SYS_RESOURCE`, `PERFMON`, `SYS_PTRACE`, and `SYS_ADMIN` capabilities.

2. **K8s-metacollector** (Deployment): A component that proxies requests to `kube-apiserver` to reduce control plane load. It also reduces the amount of metadata passed to `falco` by keeping only node-related data. K8s-metacollector collects metadata from `kube-apiserver` about Pod, Namespace, Deployment, ReplicaSet, ReplicationController, and Service resources.

   The component includes the following containers:

   - **k8s-metacollector**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to k8s-metacollector metrics.

Module interactions

The `runtime-audit-engine` module interacts with the following components:

1. **Kube-apiserver**:
   - Manages FalcoAuditRules custom resources.
   - Monitors Pod, Namespace, Deployment, ReplicaSet, ReplicationController, and Service resources.
   - Authorizes module component requests.
2. **Containerd**:
   - Provides container metadata.
   - Provides `containerd` events.
3. **Linux kernel**: Intercepts Linux kernel system calls (syscalls) in real time.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Sends webhook requests to validate the FalcoAuditRules custom resource.
   - Sends audit events.
2. **Prometheus-main**: Collects module metrics.

Additional resources

- Module `control-plane-manager` documentation
- Module `runtime-audit-engine` documentation

Secrets-store-integration module

The `secrets-store-integration` module delivers secrets to applications in Deckhouse Kubernetes Platform (DKP) from an external store compatible with the HashiCorp Vault API.

The module provides the following capabilities:

- Delivers secrets to pods as mounted files or environment variables without storing them in etcd.
- Automatically connects to an internal Deckhouse Stronghold instance without manual configuration (`DiscoverLocalStronghold` mode).
- Works with any Vault-compatible secrets store in `Manual` mode.
- Automatically refreshes mounted secrets every two minutes when the value in the store changes.
- Injects an entrypoint for applications that cannot be modified to read secrets directly from the store.
- Delivers Base64-encoded binary secrets (for example, JKS keystores or Kerberos keytab files) with automatic decoding.

The operating mode (`Manual` or `DiscoverLocalStronghold`) is specified by the `settings.connectionConfiguration` parameter in the module settings.

The module works with the following custom resources:

- SecretProviderClass: Describes which secrets to deliver to a pod and from which external store. The resource specification also defines connection parameters for the secrets source and path mappings in the container.
- SecretProviderClassPodStatus: Contains the status of the secret mounting process in a pod and diagnostic information.
- SecretsStoreImport: Stores the mapping of secrets between a Vault-compatible store and files in containers.

For more details about the module, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `secrets-store-integration` module and its interactions with other DKP components are shown in the following diagram:

Module components

The `secrets-store-integration` module consists of the following components:

1. **Csi-secrets-store** (DaemonSet): Runs on all cluster nodes and implements the Container Storage Interface (CSI) standard to deliver secrets to a pod as mounted files.

   Csi-secrets-store performs the following actions:

   - Registers the ephemeral CSI driver `secrets-store.csi.deckhouse.io` in kubelet.
   - Interacts with vault-csi-provider to retrieve data from the secrets store.
   - Mounts secrets as files in a pod.
   - Rotates secrets.
   - Watches the Pod and CSIDriver resources and the SecretProviderClass custom resource.
   - Works with the SecretProviderClassPodStatus custom resource.

   The component includes the following containers:

   - **injector-puller**: Init container that performs a one-time launch of the injector binary (an executable that securely injects secrets from an external store into the application environment inside the pod) from the `secrets-store-integration/env-injector` service image. This one-time launch is necessary to preload the image onto every cluster node. The webhook component uses this image to deliver secrets to user applications as environment variables.

   - **csi-node-driver-registrar**: Sidecar container that registers the CSI Node Plugin in kubelet. Calls the `GetPluginInfo` and `NodeGetInfo` RPCs in the secrets-store container to get plugin and node

information.
    - **secrets-store**: Main container.
    - **csi-livenessprobe**: Sidecar container that monitors the CSI driver Unix socket and exposes the `/healthz` HTTP endpoint monitored by kubelet. If the `livenessProbe` check fails, kubelet restarts the csi-secrets-store pod.
    - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to secrets-store container metrics.
2. **Vault-csi-provider** (DaemonSet): Runs on all cluster nodes and consists of a single **vault-csi-provider** container. Vault-csi-provider authenticates with the secrets store, retrieves data from it, and passes the data to csi-secrets-store.
3. **Ssi-controller** (Deployment): Consists of a single **ssi-controller** container. The controller watches SecretsStoreImport custom resources and creates SecretProviderClass custom resources based on them.
4. **Webhook** (Deployment): Implements mutating webhooks for Pod resources using Mutating Admission Controllers.

   The component mutates a pod if the pod has the `secrets-store.deckhouse.io/role` annotation. In this case, the component makes the following modifications to the pod manifest:

   - Adds an init container that copies a statically built injector binary from the `secrets-store-integration/env-injector` service image into a temporary directory shared by all containers in the pod.
   - If the pod manifest has the `secrets-store.deckhouse.io/env-from-path` annotation or a container uses secrets from the secrets store, then for each such container, including init containers, the component replaces the original startup command with a command that starts the injector binary.

     The injector receives the original startup command as an argument. If the container in the pod manifest has no startup command, the component retrieves the startup command from the image in the container registry.

   The component includes the following containers:

   - **vault-secrets-webhook**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to vault-secrets-webhook container metrics.

The following components are not part of the module, but the module affects them:

- **User-app**: Represents a typical user application that needs secrets delivered as environment variables.

  The component includes the following containers:

  - **copy-env-injector**: Init container added by the webhook component. It copies the injector binary from the `secrets-store-integration/env-injector` service image.
  - **<CONTAINER_NAME>**: One or more containers (including init containers) of the original user application whose startup command was changed by the webhook component to start the injector binary.

  The injector authenticates with the secrets store, retrieves data, passes secrets to the application as environment variables, and starts the original container command. If the `secrets-store.deckhouse.io/restart-on-secret-change` annotation is set to `watch-for-lease` or `watch-for-data`, the injector rotates secrets when they change in the store.

Module interactions

The `secrets-store-integration` module interacts with the following components:

1. **Kube-apiserver**:
   - Authorizes requests.
   - Retrieves CSIDriver, Secret, ConfigMap, and ServiceAccount.
   - Works with the SecretsStoreImport, SecretProviderClass, and SecretProviderClassPodStatus custom resources.
2. **Container registry**: Retrieves the startup command from the user container image.
3. **Secrets store**: Authorizes requests and retrieves secrets.

The following external components interact with the module:

1. **Kube-apiserver**: Invokes the mutating webhook when a Pod resource is created.
2. **Kubelet**:
   - Registers the Node Plugin.
   - Checks the CSI driver `livenessProbe`.
   - Calls the `NodePublishVolume` and `NodeUnpublishVolume` RPCs in the Node Plugin.

3. **Prometheus-main**: Collects metrics from the csi-secrets-store and webhook components.

Additional resources

- Module `secrets-store-integration` documentation

Network subsystem

This subsection describes the architecture of the Network subsystem of Deckhouse Kubernetes Platform (DKP).

The Network subsystem includes the following modules:

- `kube-dns`: Installs CoreDNS components for DNS management in the Kubernetes cluster.
- `node-local-dns`: Deploys a caching DNS server on each cluster node and exports DNS metrics to Prometheus for analyzing DNS operation in the cluster on the Grafana dashboard. The architecture of the caching DNS server is described on the corresponding page of this subsection.
- `kube-proxy`: Manages the kube-proxy components responsible for networking and load balancing within the cluster.
- `cni-cilium`: Provides cluster networking using the CNI Cilium plugin.
- `cilium-hubble`: Provides visualization of the cluster network stack if the Cilium CNI is enabled.
- `ingress-nginx`: Installs and manages the Ingress NGINX Controller using custom resources. The module architecture is described on the corresponding page of this subsection.
- `metallb`: Implements the LoadBalancer mechanism for Services in bare-metal clusters.

The subsection also describes:

- Cluster architecture with Istio enabled
- Application service architecture with Istio enabled

Cni-cilium module

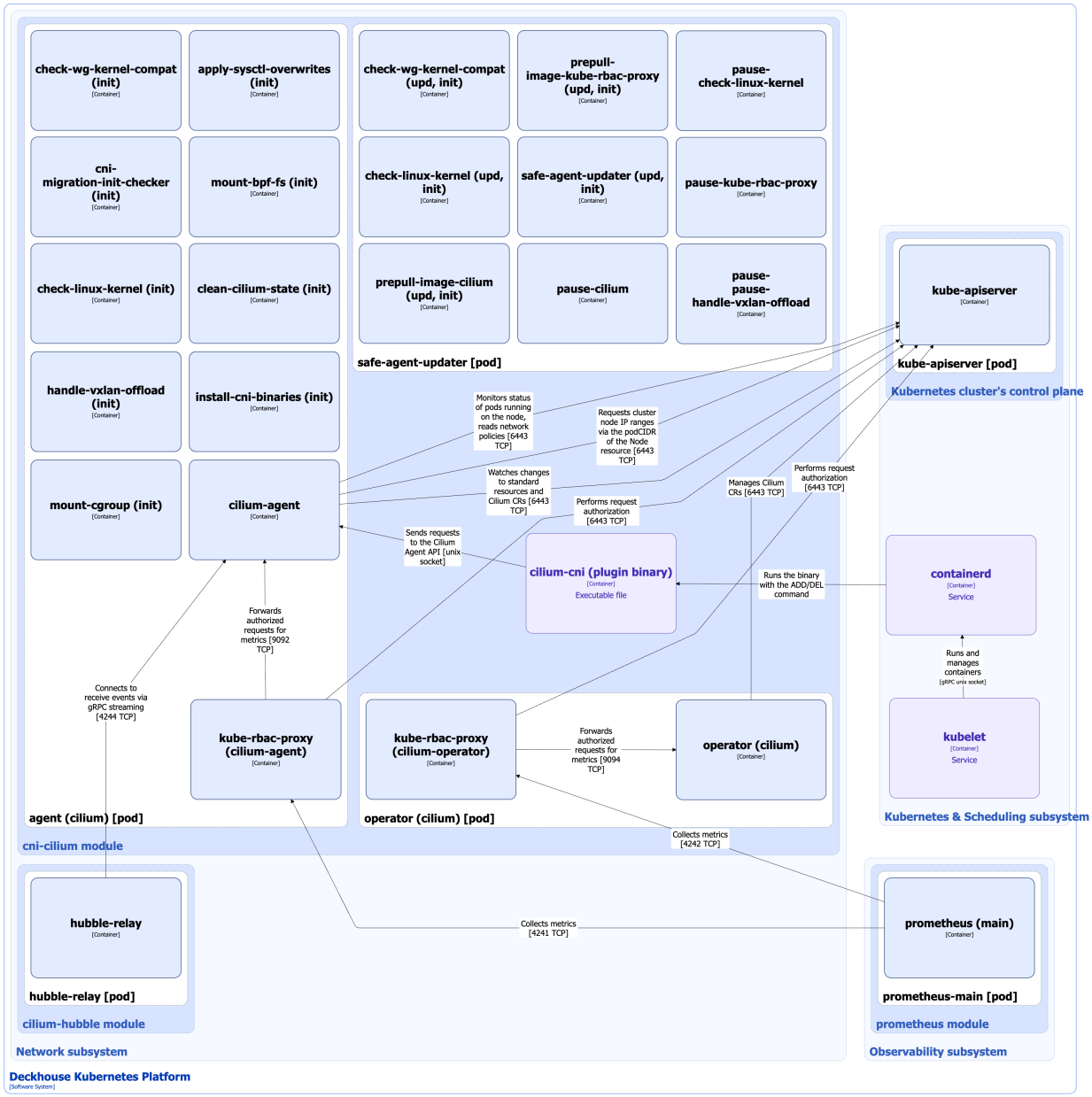
The `cni-cilium` module provides a network in a cluster. It is based on the Cilium project.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cni-cilium` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Operator** (Deployment): A component in the Cilium architecture that takes over the centralized management of global tasks in a Kubernetes cluster. Operator component performs the following functions:

   - **Custom resources management**: Automatically registers the Custom Resource Definitions (CRD) required for Cilium operation in the Kubernetes API, for example, CiliumEndpoint, CiliumEndpointSlice, CiliumNetworkPolicy. The full list of CRDs can be found in the Cilium documentation.
   - **Garbage collection for security identifiers**: Maintains a local cache of active identifiers (CiliumIdentity) and periodically scans it, deleting records about identifiers that have stopped signaling activity (heartbeat). This is important because identifiers are represented by a 16-bit integer, and running out of them (maximum 65536) can lead to problems.
   - **CiliumEndpointSlices (CES) Management**: Maintains up-to-date information about endpoints in the cluster, which is critical for the correct operation of network policies and load balancing.

   It consists of the following containers:

   - **operator**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to operator metrics. It is an open source project.

2. **Agent** (DaemonSet): Agent running on each node of the Kubernetes cluster. It is the main management component in the Cilium architecture. It is responsible for translating high-level configurations from the Kubernetes API into low-level settings for network interfaces and eBPF programs that actually handle traffic. Agent component performs the following functions:

- **Endpoints management**: Monitors the status of all endpoints on its node. Each pod is assigned a unique IP address and a security identity (numeric identifier), which is generated based on Kubernetes labels.
- **Network policies enforcement**: Calculates and applies network policies rules set in the cluster. Cilium uses the identity-based policies concept: the rules are based not on IP addresses, but on the security identity of the pod. The agent converts these rules into data structures (BPF maps) that the Linux kernel uses to filter traffic.
- **IP Address Management (IPAM)**: Allocates IP addresses for pods scheduled on its node using IP address ranges from the `spec.podCIDRs` or `spec.PodCidr` fields of the specification of the corresponding Node resource.
- **Datapath Orchestration**: Compiles, loads eBPF programs into the Linux kernel and binds them to network interfaces. It is these programs in the kernel that perform real packet processing: forwarding, policy filtering, connection tracking, NAT and load balancing.
- **Service Load Balancing**: Configures load balancing rules for services in a cluster using eBPF capabilities.
- **Cluster event handling**: The Agent works according to the event model: it constantly monitors changes in the Kubernetes API (creating pods, updating services, changing Endpoints, updating custom resources) and updates the configuration of eBPF programs on the fly.
- **Oservability**: The agent runs a local Hubble server (embedded in the agent process), which collects metrics and data on network flows. This allows you to monitor the network status, identify problems, and analyze traffic between pods. The Hubble server starts when the `cilium-hubble` module is enabled. The hubble-relay component of the `cilium-hubble` module connects to Cilium agents to receive events via a gRPC streaming.

How it works: when an event occurs in the cluster (for example, a new pod starts), the agent receives a notification about it and, based on the current cluster status and policies, configures eBPF programs so that traffic for this pod passes correctly.

It consists of the following containers:

- **cni-migration-init-checker**: Init container waiting for the migration process from another CNI plugin to complete, if such a migration has been started.
- **check-wg-kernel-compat**: Init container checking the Linux kernel version for compliance with the minimum requirements for working with WireGuard used in CNI Cilium.
- **check-linux-kernel**: Init container checking the Linux kernel version for compliance with the minimum requirements for working with CNI Cilium.
- **clearing-unnecessary-iptables**: Init container running script for cleaning up unnecessary `iptables` rules.
- **handle-vxlan-offload**: Init container correcting UDP segmentation settings (if tunnel mode is set to `VXLAN`).
- **config**: Init container generating configuration file used to configure the agent.
- **mount-cgroup**: Init container configuring Linux cgroups (control groups).
- **apply-sysctl-overwrites**: Init container configuring the Linux kernel parameters.
- **mount-bpf-fs**: Init container mounting the BPF file system.
- **clean-cilium-state**: Init container running a script for clearing the CNI Cilium state.
- **install-cni-binaries**: Init container running a script for installing binary files used by the Agent.
- **agent**: Main container.
- **kube-rbac-proxy**: Sidecar container providing authorized access to agent metrics (described above).

3. **Cilium-cni**: Executable file run by the containerd component that passes certain commands as an argument in accordance with the CNI specification, for example, ADD when starting the container and DEL when deleting it. Cilium-cni executable interacts with the Cilium agent API via a Unix socket and initiates a datapath configuration to provide network connectivity, load balancing, and network policies for pod. Datapath in Cilium is a component that runs in the Linux kernel and is responsible for real low-level processing of network packets in a Kubernetes cluster. Simply put, this is the “road” along which data travels: how packets travel from one pod to another, how they are routed, how network policies and load balancing are applied.
4. **Safe-agent-updater** (DaemonSet): A special application designed to eliminate disruptive situations that may occur when updating the version of the Cilium agent and lead to problems with the network availability of DKP components.

   It consists of the following containers:

   - **check-wg-kernel-compat**: Init container checking the Linux kernel version for compliance with the minimum requirements for working with WireGuard used in CNI Cilium.
   - **check-linux-kernel**: Init container checking the Linux kernel version for compliance with the minimum requirements for working with CNI Cilium.
   - A set of init containers for prepulling new versions of images of the corresponding containers of the agent component:
     - **prepull-image-cilium**.

        - **prepull-image-kube-rbac-proxy**.

    - **safe-agent-updater**: Init container that compares the values of special annotations in the DaemonSet manifest of the updated Cilium agent with the values of the corresponding annotations in the metadata of the agent pod running on the node. In particular, the `safe-agent-updater-daemonset-generation` annotation stores the hash sum of the agent image. If the hash sums do not match, safe-agent-updater deletes the running pod and waits until the pod with the new agent version enters the `Ready` state.

    - A set of sidecar containers for prepulling new versions of images of the corresponding containers of the agent component. Containers are on pause and perform only the function of storing images:

        - **pause-cilium**.
        - **pause-check-linux-kernel**.
        - **pause-kube-rbac-proxy**.
        - **pause-pause-handle-vxlan-offload**.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

    - Manages Cilium custom resources.
    - Watches changes to standard Kubernetes API resources and Cilium custom resources.
    - Monitors status of pods running on the node, reads network policies.
    - Requests the IP ranges of cluster nodes via the `PodCidr`/`podCIDRs` attributes of the Node resource specification.
    - Authorizes requests for agent and operator metrics.

The following external components interact with the module:

1. **Hubble-relay**: Connects to Cilium agents to receive events via gRPC streaming.
2. **Prometheus-main**: Collects metrics from agent and operator.
3. **Containerd**: Runs the cilium-cni executable file with certain commands according to the CNI specification, for example, ADD when starting the container and DEL when deleting it.

Additional resources

- Module `cilium-hubble` documentation
- Module `cni-cilium` documentation

Cilium-hubble module

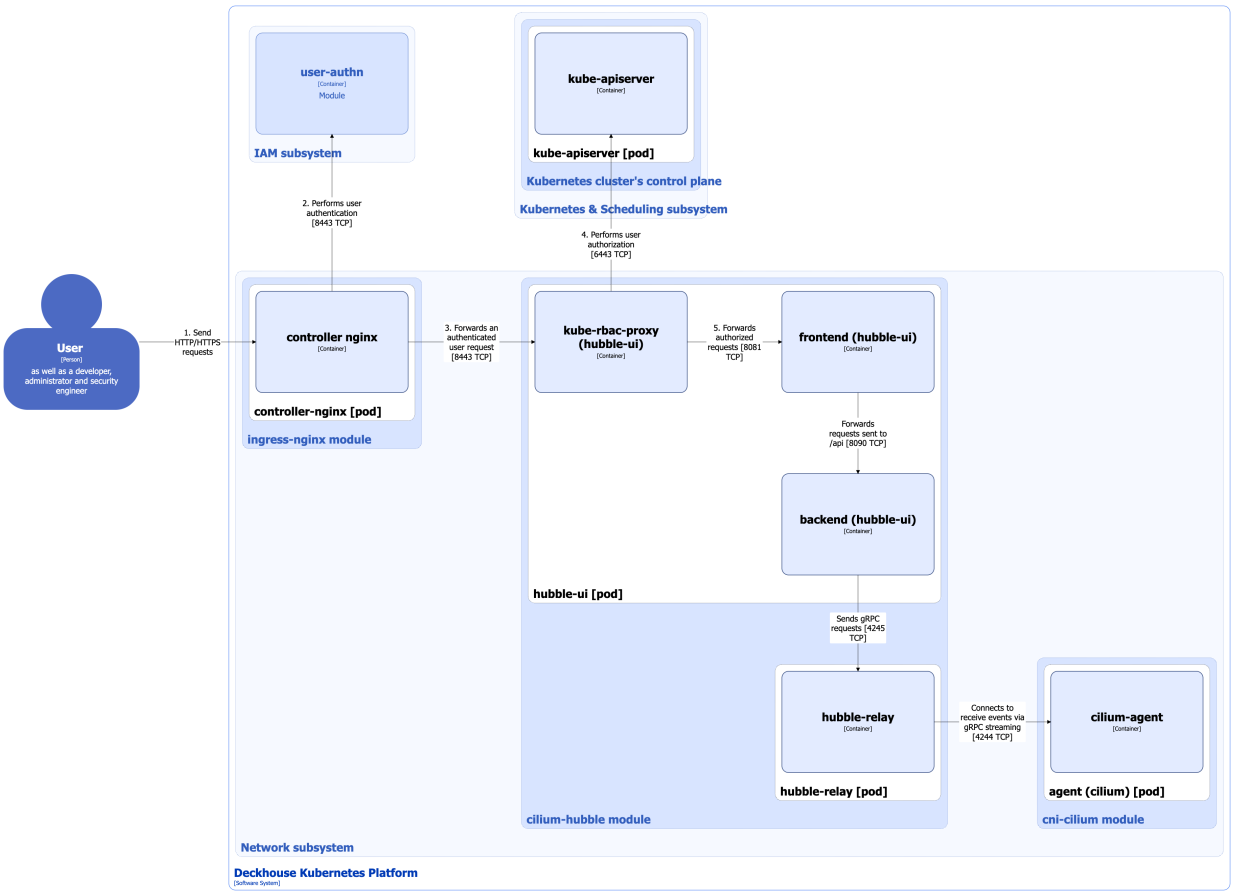
The `cilium-hubble` module provides visualization of the cluster network stack if the Cilium CNI is enabled.

For more details about module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `cilium-hubble` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



The numbers in the diagram indicate the sequence of steps that a user request goes through before reaching the hubble-ui component:

- In steps 1, 2, and 3, the request passes through the Ingress NGINX Controller, where mandatory user authentication is performed using the `user-authn` module;
- In steps 4 and 5, the user is authorized based on Kubernetes RBAC to provide secure access.

Module components

The module consists of the following components:

1. **Hubble-relay**: Component that aggregates events from all cluster nodes into a single view. Hubble-relay establishes a permanent connection to each Cilium Agent on the nodes and, through the gRPC stream, receives events, deduplicates them, and provides a single gRPC endpoint for the Hubble CLI and Hubble UI (web interface). While doing this, hubble-relay does not store history, it broadcasts an event stream in real time.

   It consists of a single container:

   - **hubble-relay**: Main container. It is a part of Cilium project.

2. **Hubble-ui**: Component that implements the web interface. Hubble-ui connects to hubble-relay and provides convenient access to data that comes from Cilium agents. Hubble-ui builds an interactive Service Map and a flow table with filters.

   It consists of the following containers:

   - **frontend**: Container that is an NGINX proxy server, which distributes static files of the hubble-ui web interface and forwards requests to the `/api` endpoint to the **backend** container of the component.
   - **backend**: Container implementing the hubble-ui API that sends requests to the hubble-relay component to receive in real time data coming from Cilium agents.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to Hubble UI. It is an open source project.

   Frontend and backend containers are built based on Hubble-UI, which is an Open Source project.

Module interactions

The module interacts with the following components:

1. **Cilium-agent**: Receives in real time an event stream from Cilium agents.
2. **Kube-apiserver**: Authorizes requests to Hubble UI.

The following external components interact with the module:

- **Controller nginx**: Forwards external user requests to the module web interface.

Additional resources

- Module `cilium-hubble` documentation

Ingress-nginx module

The `ingress-nginx` module installs and manages the Ingress NGINX Controller using the IngressNginxController custom resource.

The module can operate in high availability (HA) mode and provides flexible configuration for placing Ingress controllers on cluster nodes, as well as tuning controller behavior according to the specifics of the underlying infrastructure.

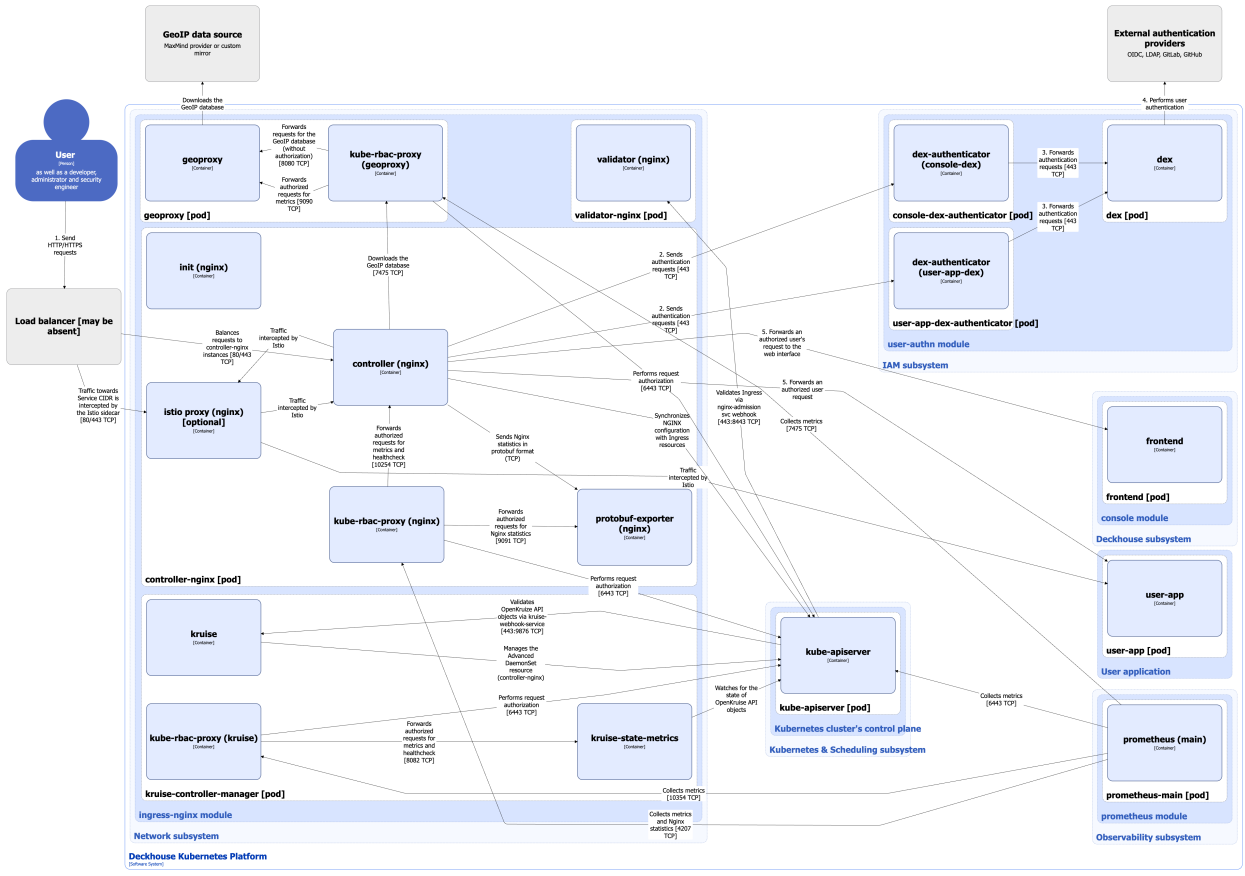
The module supports running and configuring multiple instances of the Ingress NGINX Controller independently. This allows, for example, separating external and intranet application Ingress resources.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `ingress-nginx` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller-nginx** (Advanced DaemonSet): Non-standard DaemonSet with advanced capabilities managed by kruise-controller-manager.

   It consists of the following containers:

   - **init**: Init container configuring a file system for the Ingress controller.
   - **controller**: Main container running Ingress NGINX Controller and implementing the core module logic. It is an open source project.
   - **protobuf-exporter**: Sidecar container in the ingress-controller pod that receives NGINX statistics in protobuf format. It parses and aggregates the messages according to predefined rules and exports metrics in Prometheus format. This exporter is developed by Flant.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC, providing secure access to controller metrics and status as well as to `protobuf-exporter`. It is an open source project.
   - **istio-proxy**: Istio sidecar container added to the pod when the `spec.enableIstioSidecar` parameter of the IngressNginxController custom resource is enabled. In this case, part of the user traffic passes through it.

2. **Validator-nginx** (Deployment): Consists of a single container. Validator is an Ingress NGINX Controller running in validation mode with a reduced set of privileges. It implements a webhook server used to validate Ingress resources via the Validating Admission Controllers mechanism.

3. **Kruise-controller-manager** (Deployment): Controller that manages the Advanced DaemonSet custom resource. This DaemonSet extension provides advanced update capabilities for the Ingress NGINX Controller that are not available in the standard Kubernetes DaemonSet controller.

   It consists of the following containers:

   - **kruise**: Main container running kruise-controller-manager.
   - **kruise-state-metrics**: Sidecar container that monitors the state of OpenKruise API objects and exposes corresponding metrics (but not metrics of the kruise-controller-manager itself).
   - **kube-rbac-proxy**: Sidecar container providing authorized access to controller metrics and status (described above).

4. **Geoproxy** (StatefulSet): Caching proxy server for the Ingress controller that provides quick access to the GeoIP database downloaded from the MaxMind provider. This component also grants access to already downloaded bases in clusters that have no internet access, improving the stability of the Ingress NGINX Controller when working with GeoIP databases.

   This component provides the following features:

   - MaxMind license saving (databases are downloaded from a single point once a day).
   - Persistent data storage (if components are restarted, it doesn’t require accessing the MaxMind servers again).
   - It lets you specify a custom mirror for downloading databases.

   It consists of the following containers:

   - **geoproxy**: Proxy server. This server is developed by Flant.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to geoproxy metrics (the GeoIP databases are available without authorization). This container is described in detail above.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Synchronizes the NGINX configuration when Ingress resources change.
   - Authorizes requests for metrics, statistics, and controller status checks.
   - Forwards external HTTP requests to the Kubernetes API endpoint.

2. **GeoIP database source** (MaxMind provider or custom mirror): The module downloads the GeoIP database.

3. **Dex-authenticator of platform services and user applications**: Used to authenticate requests in dex via dex-authenticator, which acts as an OAuth2 proxy.

4. **DKP platform services** (such as `console`, `dashboard`, Grafana, and others): The module forwards HTTP requests that have been authenticated via Dex.

5. **User services deployed in DKP**: The module forwards external HTTP requests to user services. To enable this, the user must create the corresponding Ingress resources and, if authentication via Dex is required, the DexAuthenticator custom resource.

> **i** To keep the diagram simple, it shows interactions between ingress-controller and only one DKP service, the frontend component of the `console` module and its corresponding console-dex-authenticator.

The following external components interact with the module:

1. **Kube-apiserver**: Uses a validation webhook to validate created or updated Ingress resources.
2. **Prometheus-main**: Collects metrics from ingress and kruise controllers, geoproxy, as well as NGINX statistics.
3. **Load balancer**: Distributes HTTP/HTTPS traffic across healthy ingress-controller instances.

Receiving traffic from external networks

Methods for receiving traffic from external networks are described in the `spec.inlet` parameter of the IngressNginxController custom resource.

For inlet types LoadBalancer, LoadBalancerWithProxyProtocol, and LoadBalancerWithSSLPassthrough, the load balancer shown in the diagram is automatically provided by the cloud provider (when DKP is deployed in a cloud environment) or can be implemented using the MetalLB controller (for bare-metal installations). For configuration details, refer to the `metallb` module documentation.

For inlet types HostPort, HostPortWithProxyProtocol, HostPortWithSSLPassthrough, and HostWithFailover, the load balancer is deployed by the user or may be absent. In this case, the user must configure the load balancer backends or otherwise ensure network connectivity to the ingress-controller. The ingress-controller entry point in this case is the ports on the cluster nodes where the controller is running.

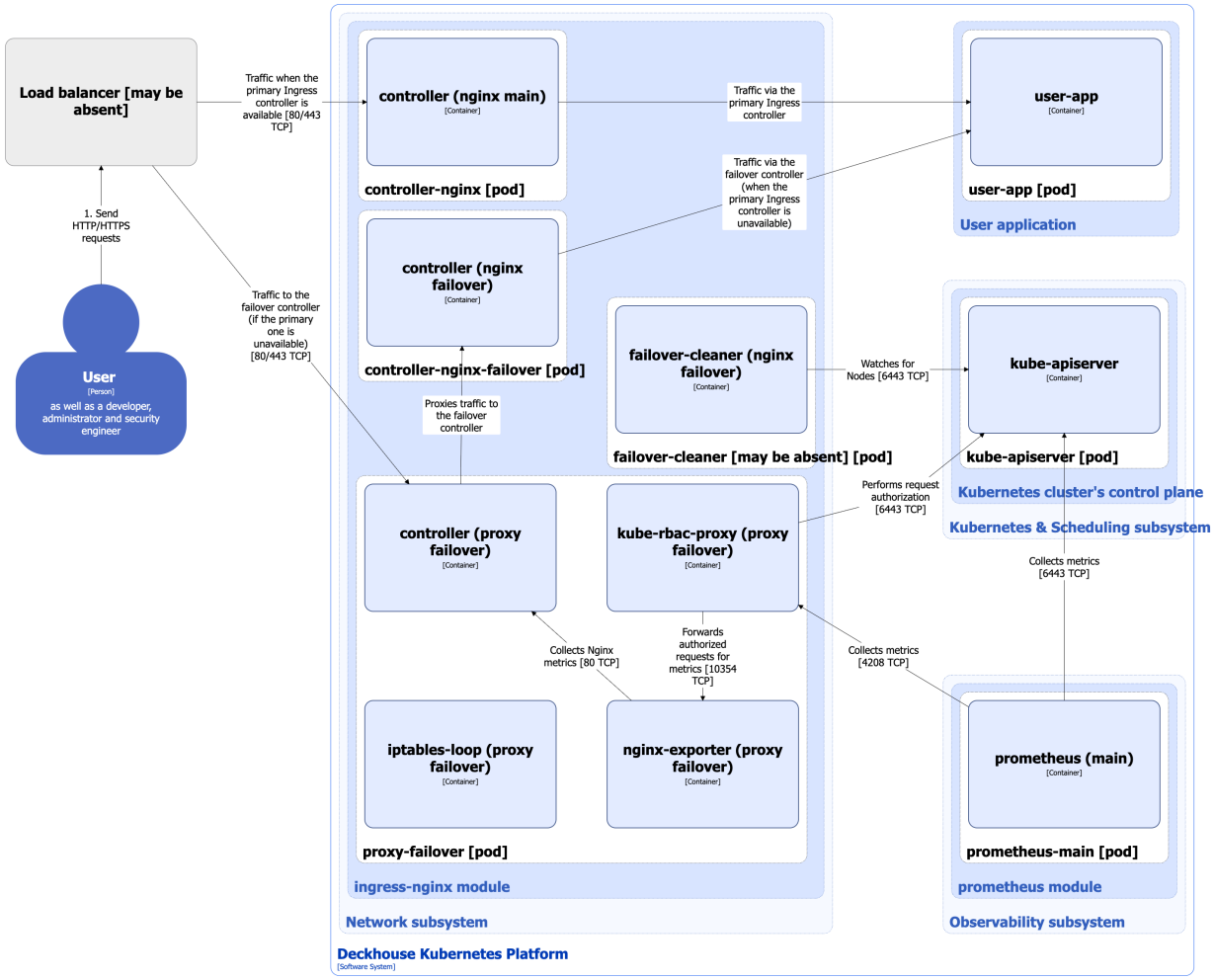
Architecture of Ingress controller with HostWithFailover inlet type

When the value `HostWithFailover` of the `spec.inlet` parameter of the IngressNginxController custom resource is set, two Ingress controllers are installed in the cluster: the primary controller and the failover controller, as well as a proxy-failover controller that coordinates traffic switch between them.

The primary controller starts in `hostNetwork`, while the failover controller starts in `podNetwork`. If the main controller pod is unavailable on the node, the proxy failover starts proxying traffic to the failover controller pod using the `PROXY PROTOCOL` to save information about the client's IP address.

> **i** The following diagram does not show the architecture of the main Ingress controller, as well as the module interactions, as they are described in detail in the diagram above.

The Level 2 C4 architecture of Ingress controller with HostWithFailover inlet type and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Components of failover Ingress controller

1. **Controller-nginx-failover** (Advanced DaemonSet): Failover Ingress-controller deployed on the same nodes as the main one. Failover controller has the same containers on the pod with the same purpose as the main Ingress controller.
2. **Proxy-failover** (Advanced DaemonSet): Proxy server.

   It consists of the following containers:

   - **controller**: Server that performs the following functions:
     - Monitors a performance of the main Ingress controller and returns its status.
     - Monitors the NGINX configuration file on the node and reboots Ingress controller if this configuration is changed.
     - Starts a NGINX instance that proxies traffic to the failover controller in case the main one is unavailable. Traffic is switched to the failover controller and back to the main controller with the use of `iptables` rules, that are configured by the iptables-loop container depending on the availability of the `80` and `443` TCP ports on the main controller.

     This controller is developed by Flant.
   - **iptables-loop**: Sidecar container that updates the `iptables` rules on the node that are necessary for the Ingress controller to work. It is developed by Flant.
   - **nginx-exporter**: Sidecar container that connects to NGINX over HTTP and exports metrics in the Prometheus format. It is an open source project.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to controller metrics and status (described above).
3. **Failover-cleaner** (DaemonSet): Component deployed to the cluster nodes that are labeled with `ingress-nginx-controller.deckhouse.io/need-hostwithfailover-cleanup=true`. It cleans up the `iptables` rules. If the Ingress controller operates normally, the failover-cleaner component doesn't run on any node.

Failover Ingress controller interactions

Failover Ingress controller interacts with the following components:

1. **Kube-apiserver**:
   - Watches Node resources for `ingress-nginx-controller.deckhouse.io/need-hostwithfailover-cleanup=true` labels.
   - Authorizes requests for metrics, statistics, and controller status checks.
   - Forwards external HTTP requests to the Kubernetes API endpoint.
2. **GeoIP database source** (MaxMind provider or custom mirror): Failover Ingress controller downloads the GeoIP database.
3. **Dex-authenticator of platform services and user applications**: Used to authenticate requests in dex via dex-authenticator, which acts as an OAuth2 proxy.
4. **DKP platform services** (such as `console`, `dashboard`, Grafana, and others): The module forwards HTTP requests that have been authenticated via Dex.
5. **User services deployed in DKP**: The module forwards external HTTP requests to user services. To enable this, the user must create the corresponding Ingress resources and, if authentication via Dex is required, the DexAuthenticator custom resource.

The following external components interact with the failover Ingress controller:

1. **Prometheus-main**: Collects failover Ingress controller metrics.
2. **Load balancer**: Routes HTTP/HTTPS traffic in case the main Ingress controller is unavailable.

Additional resources

- Module `ingress-nginx` documentation
- Module `metallb` parameters
- Module `user-authn` custom resources

Metallb module

The `metallb` module implements the LoadBalancer mechanism for services in bare-metal clusters.

The following operating modes are supported:

- **Layer 2**: Implements an improved load balancing mechanism (compared to the standard L2 mode in MetalLB) for bare-metal clusters. It allows using multiple "public" IP addresses for services and distributing them evenly across available cluster nodes.

- **BGP**: Fully based on the MetalLB solution and uses the BGP protocol to create routes.

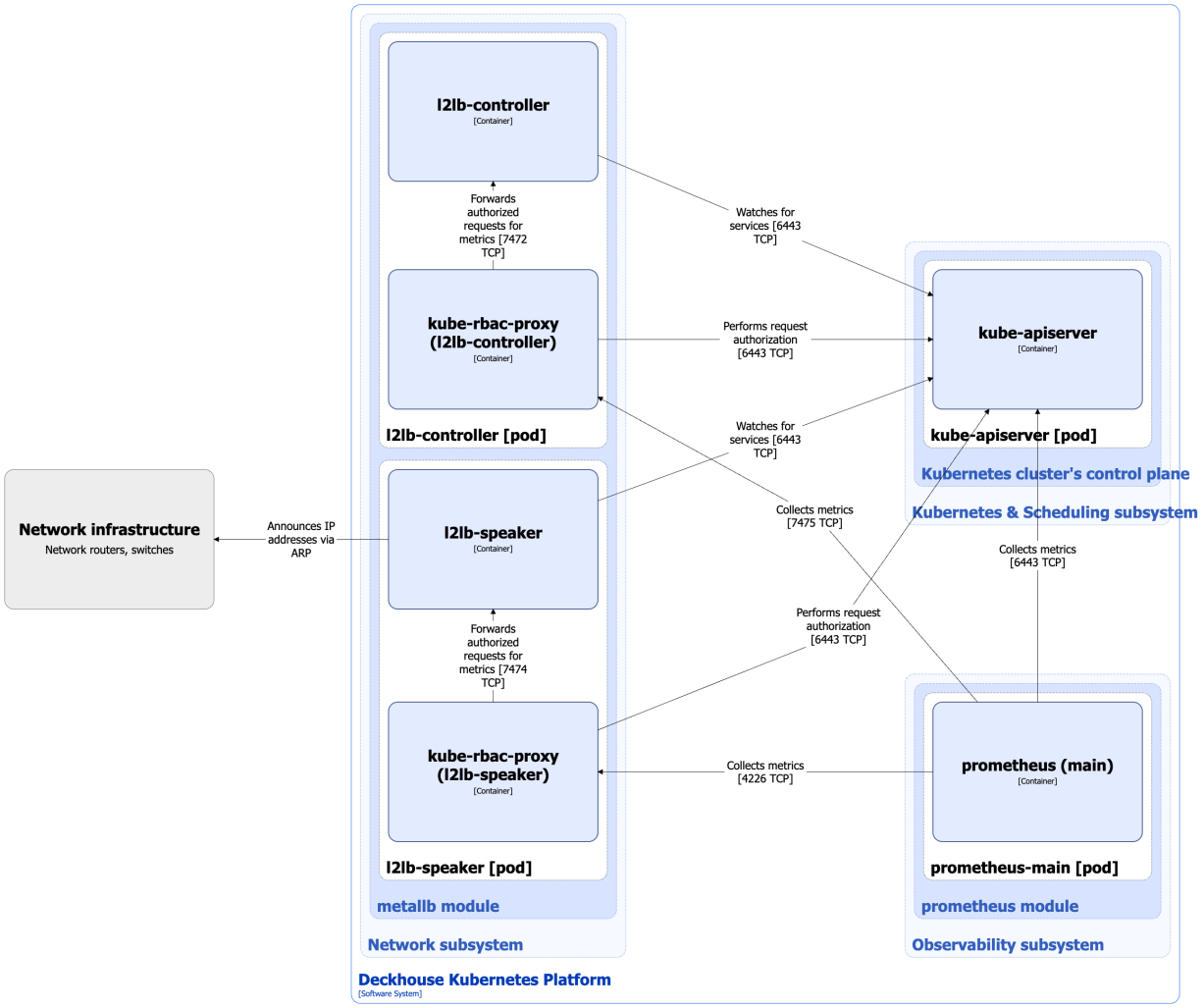
For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

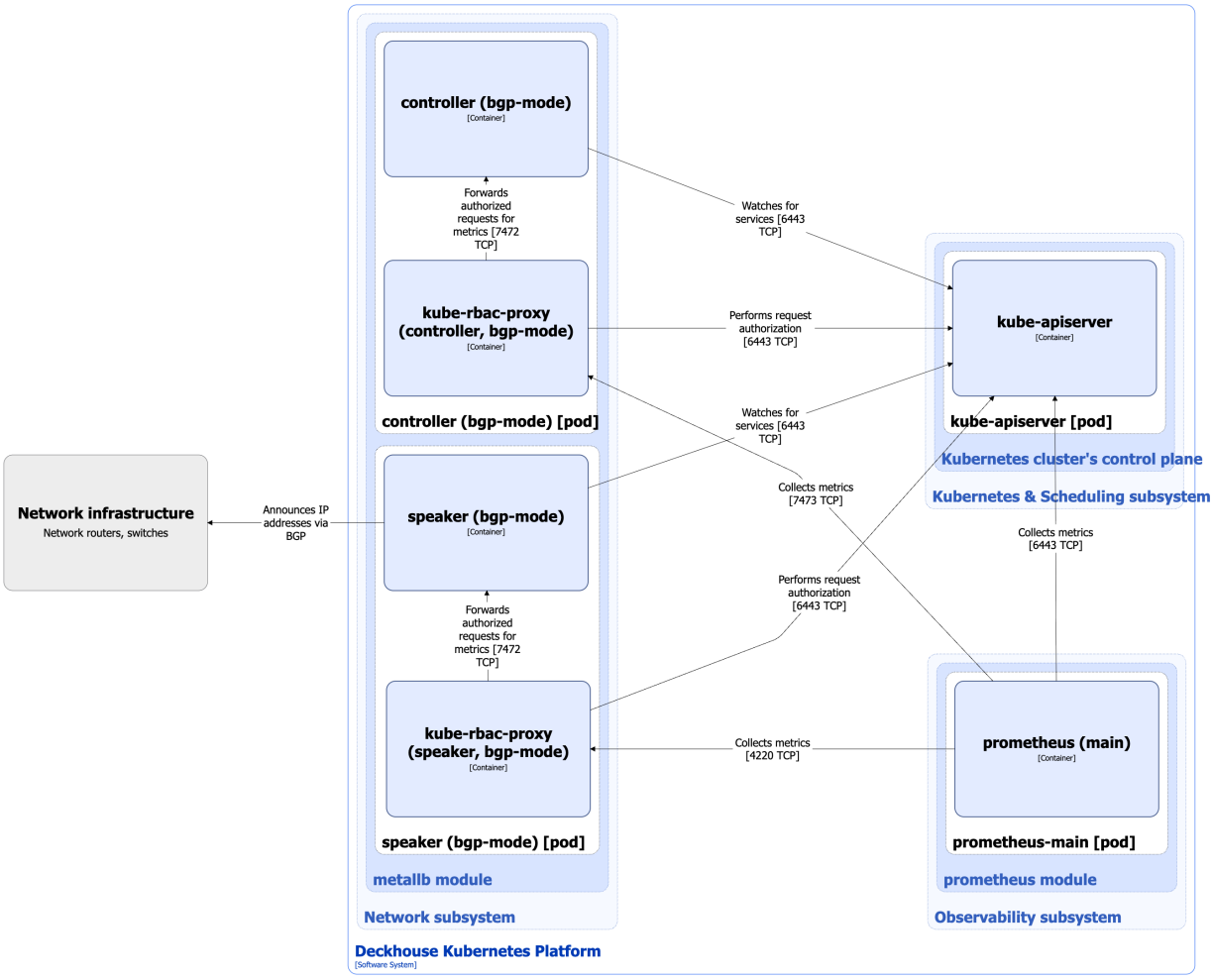
> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `metallb` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagrams.

MetalLB in Layer 2 mode:



MetalLB in BGP mode:



Module components

The module consists of the following components:

1. **Controller/l2lb-controller** (Deployment): MetalLB controller responsible for assigning IP addresses to LoadBalancer services.

   The controller monitors changes in Kubernetes services and applies IP address configuration based on a predefined address pool specified in the module configuration.

   It consists of the following containers:

   - **controller/l2lb-controller**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to controller metrics. It is an open-source project.

2. **Speaker/l2lb-speaker** (DaemonSet): MetalLB speaker running on each cluster node that participates in load balancing. It is responsible for implementing the load balancing mechanism at the network level.

   Depending on the operating mode, it performs the following functions:

   - In **Layer 2 mode**, it uses ARP to associate virtual IP addresses of services with the physical addresses of cluster nodes.
   - In **BGP mode**, it announces routes to virtual IP addresses via the BGP protocol, making services accessible from outside the cluster.

   It consists of the following containers:

   - **speaker/l2lb-speaker**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to speaker metrics.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Monitors changes in Kubernetes services and applies IP address configuration.
   - Authorizes requests for metrics.

2. **Network equipment**: Ensures the availability of virtual IP addresses outside the cluster:
    - In **BGP mode**, it advertises routes to virtual IP addresses using the BGP protocol.
    - In **Layer 2 mode**, it uses ARP/GARP to notify upstream routers that the virtual IP addresses are associated with the MAC addresses of specific nodes.

The following external components interact with the module:

- **Prometheus-main**: Collects metrics from the MetalLB controller and speaker.

Additional resources

- Module `metallb` documentation

DNS modules

The `kube-dns` module provides domain name resolution based on CoreDNS in Deckhouse Kubernetes Platform (DKP).

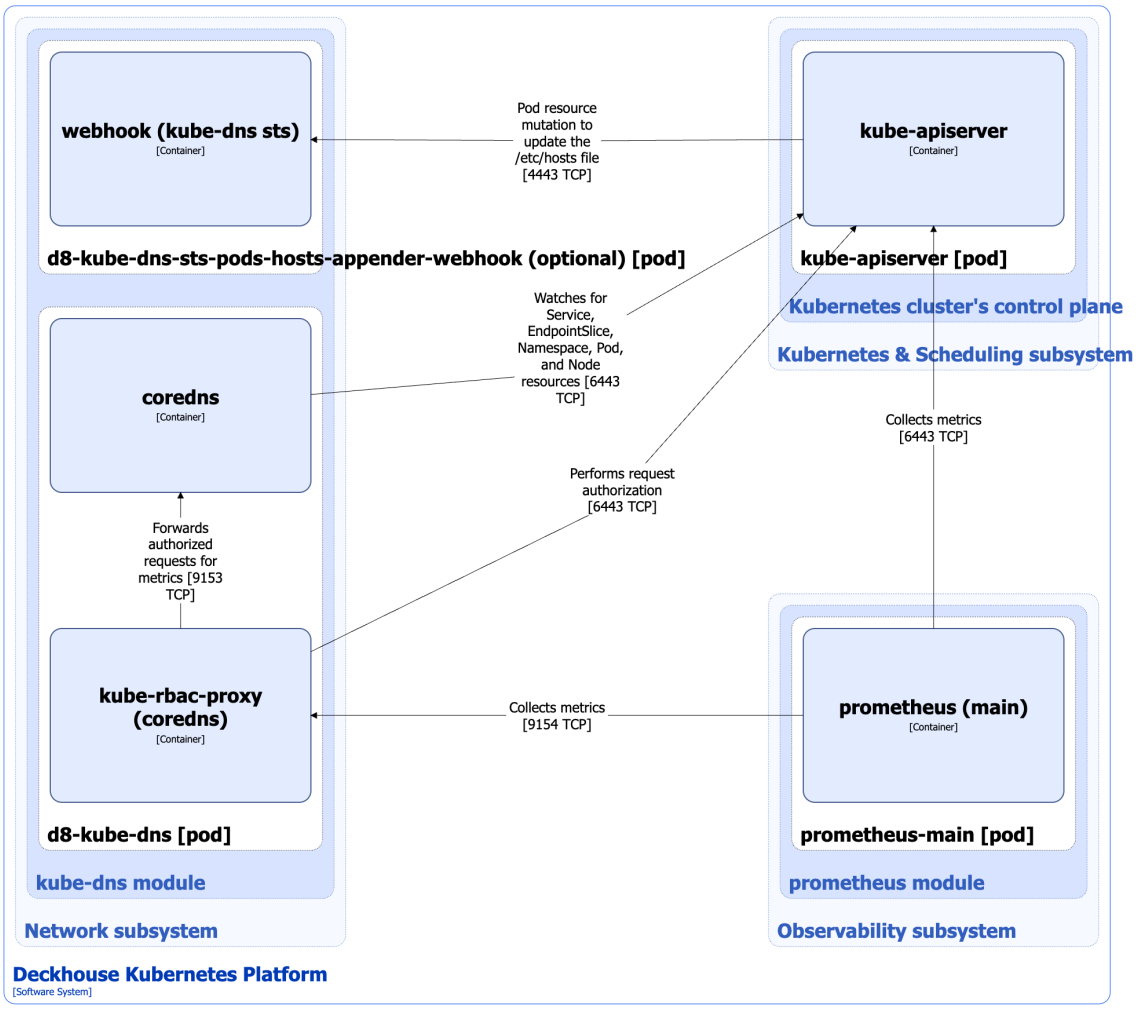
For more details about module configuration and usage examples, refer to the corresponding documentation section.

kube-dns module

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `kube-dns` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The `kube-dns` module consists of the following components:

1. **D8-kube-dns** (Deployment): Main module component implementing a DNS server in the Kubernetes cluster.

   The d8-kube-dns component watches changes in standard Service, EndpointSlice, Namespace, and Pod resources, and periodically requests Node resources. Based on the data it receives, d8-kube-dns updates records in the local object database.

   It consists of the following containers:

   - **coredns**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to coredns metrics. It is an Open Source project.

2. **D8-kube-dns-sts-pods-hosts-appender-webhook** (Deployment): Optional component that consists of a single **webhook** container.

   The Deckhouse controller of the `deckhouse` module creates this component if the `.spec.settings.clusterDomainAliases` parameter is set in ModuleConfig.

   The component implements a mutating webhook server that adds the **render-etc-hosts-with-cluster-domain-aliases** init container to a pod created by a StatefulSet controller if the `.spec.subdomain` option is specified in the pod spec.

   The init container modifies the `/etc/hosts` file so that the name resolution subsystem works correctly with cluster domain aliases.

Module interactions

The module interacts with the following components:

- **Kube-apiserver**:
  - Watches standard Service, Endpoint, EndpointSlice, Namespace, Pod, and Node resources.
  - Authorizes requests for metrics.

The following external components interact with the module:

1. **Kube-apiserver**: Modifies Pod resources created by the StatefulSet controller.
2. **Prometheus-main**: Collects module metrics.

node-local-dns module

The `node-local-dns` module provides a caching DNS service and a DNS request forwarding mechanism on each cluster node, reducing the load on `coredns` of the `kube-dns` module.

Depending on the CNI plugin used, the `node-local-dns` module implements different DNS request forwarding mechanisms:

- When using `cni-cilium`, Cilium handles DNS request forwarding using the CiliumLocalRedirectPolicy custom resource.
- When using `cni-flannel` or `cni-simple-bridge`, iptables handles DNS request forwarding.

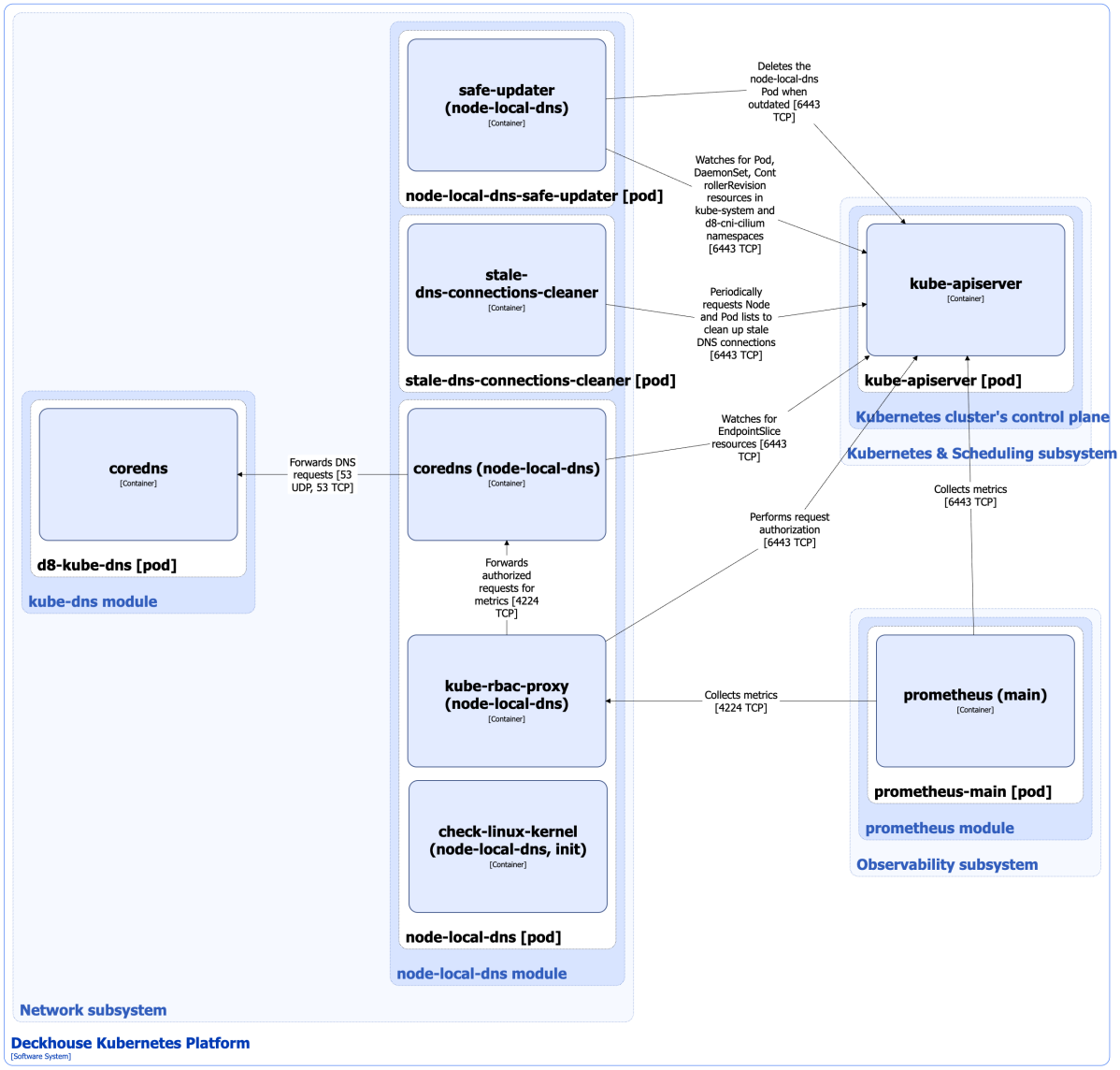
For more details about module configuration and usage examples, refer to the corresponding documentation section.

When using cni-cilium

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `node-local-dns` module when using Cilium as the CNI plugin and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The `node-local-dns` module consists of the following components:

1. **Node-local-dns** (DaemonSet): Main module component implementing a caching DNS server in the Kubernetes cluster.

   The node-local-dns component watches changes in standard EndpointSlice resources and uses them to update the list of DNS servers for forwarding requests.

   It consists of the following containers:

   - **check-linux-kernel**: Init container that checks the Linux kernel version.
   - **coredns**: Main container.
   - **cube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to coredns metrics. It is an Open Source project.

2. **Stale-dns-connections-cleaner** (DaemonSet): Component that removes stale UDP connections left after the `node-local-dns` Pod is restarted. It consists of a single **stale-dns-connections-cleaner** container.

   > ⚠ The component has privileged access to the network subsystem of each node. On Linux, this requires the `CAP_NET_ADMIN` capability. This access is required to perform network connection operations at the Linux kernel level.

3. **Safe-updater** (Deployment): Component that provides safe restarts of node-local-dns when the DaemonSet spec changes.

   Safe-updater checks that Cilium is running on the node and is in a correct state, and only then sends a command to delete the `node-local-dns` Pod.

Module interacts

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches standard EndpointSlice, DaemonSet, ControllerRevision, and Pod resources.
   - Periodically retrieves Node resources.
   - Deletes the `node-local-dns` Pod when the DaemonSet configuration becomes outdated.
   - Authorizes requests for metrics.
2. **D8-kube-dns**: Executes DNS requests.

The following external components interact with the module:

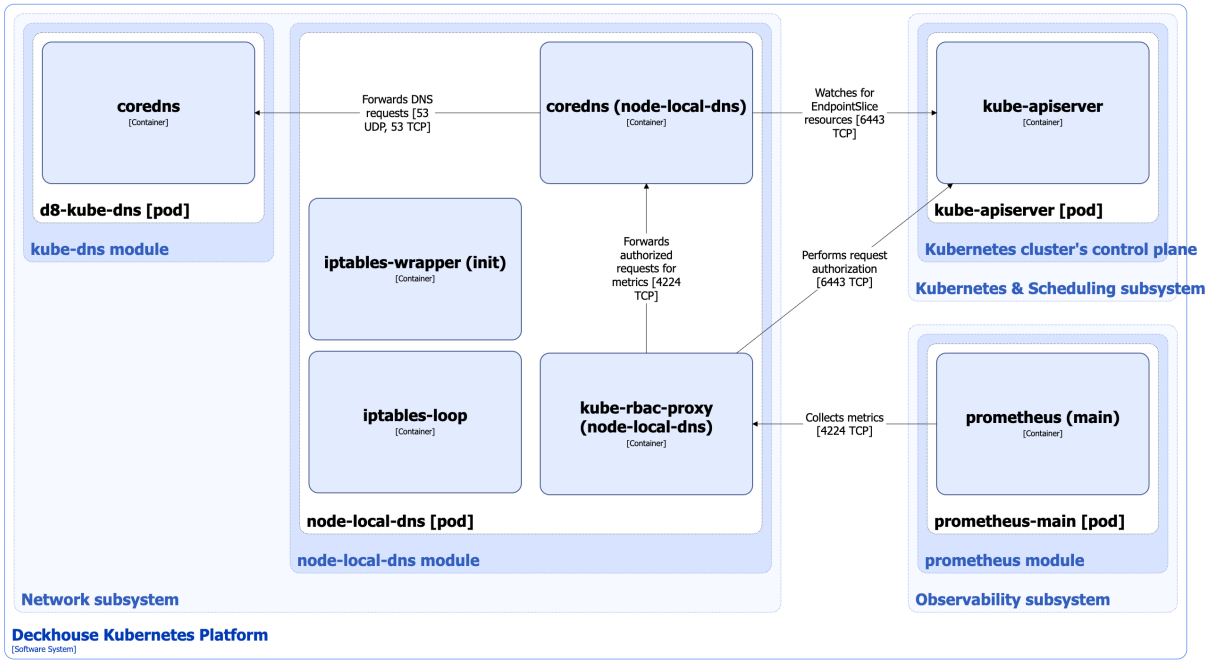
- **Prometheus-main**: Collects module metrics.

When using cni-flannel or cni-simple-bridge

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `node-local-dns` module when using the `cni-flannel` or `cni-simple-bridge` CNI plugin and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The `node-local-dns` module consists of the following components:

- **Node-local-dns** (DaemonSet): Main module component implementing a caching DNS server in the Kubernetes cluster.

  The node-local-dns component watches changes in standard EndpointSlice resources and uses them to update the list of DNS servers for forwarding requests.

  It consists of the following containers:

  - **iptables-wrapper**: Init container that prepares executables required for working with iptables.
  - **coredns**: Main container.
  - **iptables-loop**: Sidecar container that updates iptables rules for DNS request forwarding based on node-local-dns readiness.
  - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to coredns container metrics. It is an Open Source project.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

    - Watches standard EndpointSlice resources.
    - Authorizes requests for metrics.

2. **D8-kube-dns**: Executes DNS requests.

The following external components interact with the module:

- **Prometheus-main**: Collects module metrics.

Additional resources

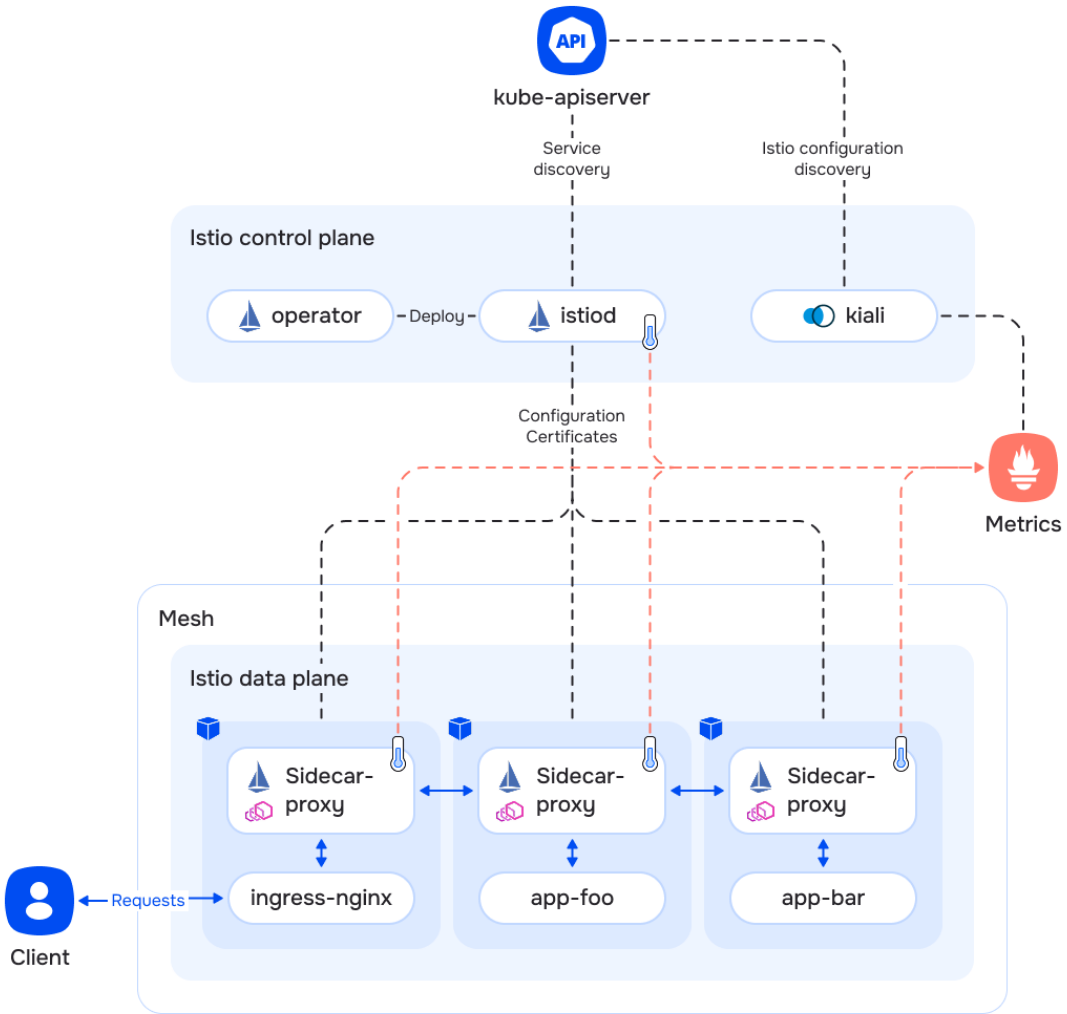
- Module `kube-dns` parameters
- Module `node-local-dns` parameters
- Caching DNS server in a cluster
- Module `deckhouse` documentation
- Module `kube-dns` documentation

Cluster-level Istio architecture

> Compatibility between Istio versions and Kubernetes versions is defined by the compatibility table in the `istio` module documentation. Before using Istio, refer to the current versions and their status in that section.

The Istio components are divided into two categories:

- **Control plane**: Managing and maintaining services. The control plane usually refers to istiod Pods.
- **Data plane**: Istio application part composed of a set of sidecar-proxy containers.



All data plane services are grouped into a service mesh with the following features:

- It has a common namespace for generating service ID in the following format: `<TrustDomain>/ns/<Namespace>/sa/<ServiceAccount>`. Each service mesh has a TrustDomain ID (in this case, it matches the cluster domain), for example, `mycluster.local/ns/myns/sa/myapp`.
- Authentication of services within a single service mesh using trusted root certificates.

Control plane components:

- **Istiod**: Main service working through the following tasks:
  - Continuous connection to the Kubernetes API and collecting information about services.
  - Processing and validating all Istio-related custom resources using the Kubernetes Validating Webhook mechanism.
  - Configuring each sidecar proxy individually:
    - Generating authorization, routing, balancing rules, etc.
    - Distributing information about other application services in the cluster.
    - Issuing individual client certificates for implementing Mutual TLS. These certificates are unrelated to the certificates that Kubernetes uses for its own service needs.
  - Automatic tuning of manifests that describe application Pods via the Kubernetes Mutating Webhook mechanism:
    - Injecting an additional sidecar-proxy service container.
    - Injecting an additional init container for configuring the network subsystem (configuring DNAT to intercept application traffic).
    - Routing readiness and liveness probes through the sidecar-proxy.
- **Operator**: Installs all the resources required to operate a specific version of the control plane.
- **Kiali**: Dashboard for monitoring and controlling Istio resources as well as user services managed by Istio:
  - Visualizing relationships between services.
  - Diagnosing problematic relationships.
  - Diagnosing control plane health.

With Istio enabled, the Ingress controller behavior is changed as follows:

- A sidecar proxy is added to the controller Pods. It only handles traffic from the controller to the application services (the `enableIstioSidecar` parameter of the IngressNginxController resource).
- Services not managed by Istio continue to function as before, requests to them are not intercepted by the controller sidecar.
- Requests to services running under Istio are intercepted by the sidecar and processed according to Istio rules. For details on enabling Istio for applications, refer to the corresponding documentation page.

The istiod controller and sidecar-proxy containers export their own metrics that the cluster-wide Prometheus collects.

Estimating overhead

Using Istio will incur additional resource costs for both control plane (istiod controller) and data plane (istio sidecars).

Control plane

The istiod controller continuously monitors the cluster configuration, compiles the settings for the data plane istio sidecars and distributes them over the network. Therefore, the more applications and their instances, the more services, and the more frequently this configuration changes, the more computational resources are required and the greater the load on the network.

Two approaches are supported to reduce the load on controller instances:

- Horizontal scaling (the `controlPlane.replicasManagement` parameter of the `istio` module). The more controller instances, the fewer instances of istio sidecars to serve for each controller and the less CPU and network load.
- Data plane segmentation using the Sidecar resource (recommended approach). The smaller the scope of an individual istio sidecar, the less data in the data plane needs to be updated and the less CPU and network load.

A rough estimate of overhead for a control plane instance that serves 1000 services and 2000 istio sidecars is 1 vCPU and 1.5 GB RAM.

Data plane

The consumption of data plane resources (istio sidecars) is affected by many factors:

- Number of connections
- The intensity of requests
- Size of requests and responses
- Protocol (HTTP/TCP)
- Number of CPU cores
- Complexity of the service mesh configuration

A rough estimate of the overhead for an istio sidecar instance is 0.5 vCPU for 1000 requests/sec and 50 MB RAM.

Istio sidecars also increase latency in network requests, about 2.5 ms per request.

Additional resources

- Module `ingress-nginx` custom resources
- Module `istio` documentation

Application service architecture with Istio enabled

> **i** Compatibility between Istio versions and Kubernetes versions is defined by the compatibility table in the `istio` module documentation. Before using Istio, refer to the current versions and their status in that section.

Architecture features

- **Sidecar proxy**: Each application service Pod gets a sidecar container, a sidecar proxy, with two applications:
  - **Envoy**: Proxies application traffic. It is responsible for implementing all the Istio functionality, including

routing, authentication, authorization, etc.
    - **Pilot-agent**: A part of Istio that is responsible for keeping the Envoy configuration up to date and has a built-in caching DNS server.
- **DNAT settings**:
    - DNAT of incoming and outgoing application requests in the sidecar proxy is configured in each Pod using an additional init container. Thus, traffic will be intercepted transparently for applications.
    - Since incoming traffic is redirected to the sidecar proxy, this also applies to readiness/liveness probes. Since the Kubernetes subsystem does not support Mutual TLS probes, all existing probes are redirected to a port in the sidecar proxy, which passes them to the application unchanged.
- **Ingress Controller**:
    - Each Pod of an Ingress controller also includes a sidecar proxy, which handles traffic between the controller and services.
    - Incoming traffic from users is handled directly by the controller.
- **Ingress resources**: These resources require minimal modification in the form of adding annotations:
    - `nginx.ingress.kubernetes.io/service-upstream: "true"`: The Ingress controller will use the service's ClusterIP as upstream instead of the Pod addresses. In this case, traffic balancing between the Pods is handled by the sidecar proxy. Use this option only if your service has a ClusterIP.
    - `nginx.ingress.kubernetes.io/upstream-vhost: "myservice.myns.svc"`: The Ingress controller's sidecar proxy makes routing decisions based on the `Host` header. If this annotation is omitted, the controller will leave a header with the site address (for example, `Host: example.com`).
- **Services**:
    - Service resources do not require modifications and continue to work without adaptation. Applications can still access service addresses like `servicename`, `servicename.myns.svc`, etc.
- **DNS queries**:
    - Internal DNS queries of Pods are transparently redirected to the sidecar proxy for processing to resolve DNS names of services from neighboring clusters.

User request lifecycle

Application with Istio disabled

Application with Istio enabled

Additional resources

- Module `istio` documentation

Caching DNS server in a cluster

Standard DNS operation in Kubernetes comes with a series of issues that may cause degradation of key service performance indicators:

- By default, the Linux kernel does not cache DNS requests, so there is no local cache in Pods.
- All DNS requests from a container result in a network request to the cluster DNS. This is also true for requests to resources located on the same node.
- A Pod's DNS request is first resolved through cluster DNS zones, and only afterward sent to external DNS servers. For example, a request to `ya.com` will first be resolved through cluster zones like `cluster.local`, `svc.cluster.local`, and `<namespace>.svc.cluster.local`. Only after receiving negative responses (meaning, on the second attempt or later) it will be resolved correctly.

Any minor network delays can significantly degrade service quality due to the above-mentioned issues.

One possible solution is to install a DNS server on each node. In Deckhouse Kubernetes Platform, this is implemented using the `node-local-dns` module.

When using a caching DNS server, external requests (that are not already cached) will still be attempted to resolve through the internal zone chain first. Under high load (for example, with many repeated requests per second for the same records, which is common), caching is enough to significantly improve DNS resolution performance.

Caching DNS server operation principles

When the caching DNS server is deployed, the `node-local-dns` module applies the following configuration steps on each cluster node:

- Configuring an interface with the IP address of the `kube-dns` service's clusterIP.
- Starting a caching CoreDNS that listens on that address.
- Adding an iptables rule: if the socket is open, traffic is redirected to it. Otherwise, a standard Kubernetes routing via ClusterIP is used:

  ```
  -A PREROUTING -d <kube-dns IP address> -m socket --nowildcard -j NOTRACK
  ```

CoreDNS configuration aspects

Key characteristics of the CoreDNS configuration:

- All requests are cached.
- All DNS requests are forwarded to the cluster DNS ClusterIP.

Grafana dashboard

The `Kubernetes / DNS (node local)` dashboard displays:

- General charts (providing an overall view of DNS performance),
- Per-node charts (helping investigate node-specific issues identified in the general charts),
- Upstream charts (helping evaluate performance of the cluster DNS and node DNS servers specified in `/etc/resolv.conf`).

Additional resources

- Module `node-local-dns` documentation

Storage subsystem

This subsection describes the architecture of the Storage subsystem of Deckhouse Kubernetes Platform (DKP).

The Storage subsystem includes the following modules:

- `local-path-provisioner`: Provides the local storage on Kubernetes nodes using `HostPath` volumes and creates StorageClass resources to manage the allocation of local storage.
- `snapshot-controller`: Enables snapshot support for compatible CSI-drivers in the Kubernetes cluster.
- `sds-local-volume`: Manages the local block storage based on LVM and enables creating StorageClass resources in Kubernetes using the LocalStorageClass resource.
- `sds-node-configurator`: Manages block devices and LVM on Kubernetes cluster nodes through Kubernetes custom resources.
- `sds-replicated-volume`: Manages replicated block storage based on DRBD. LINSTOR is used as a control plane.
- `storage-volume-data-manager`: Provides secure export and import of persistent volume contents over HTTP protocol.

- Modules that provide a CSI driver implementation for integration with various types of storage (software and hardware):
  - `csi-ceph`
  - `csi-hpe`
  - `csi-huawei`
  - `csi-netapp`
  - `csi-nfs`
  - `csi-s3`
  - `csi-scsi-generic`
  - `csi-vsphere`
  - `csi-yadro-tatlin-unified`

Local-path-provisioner module

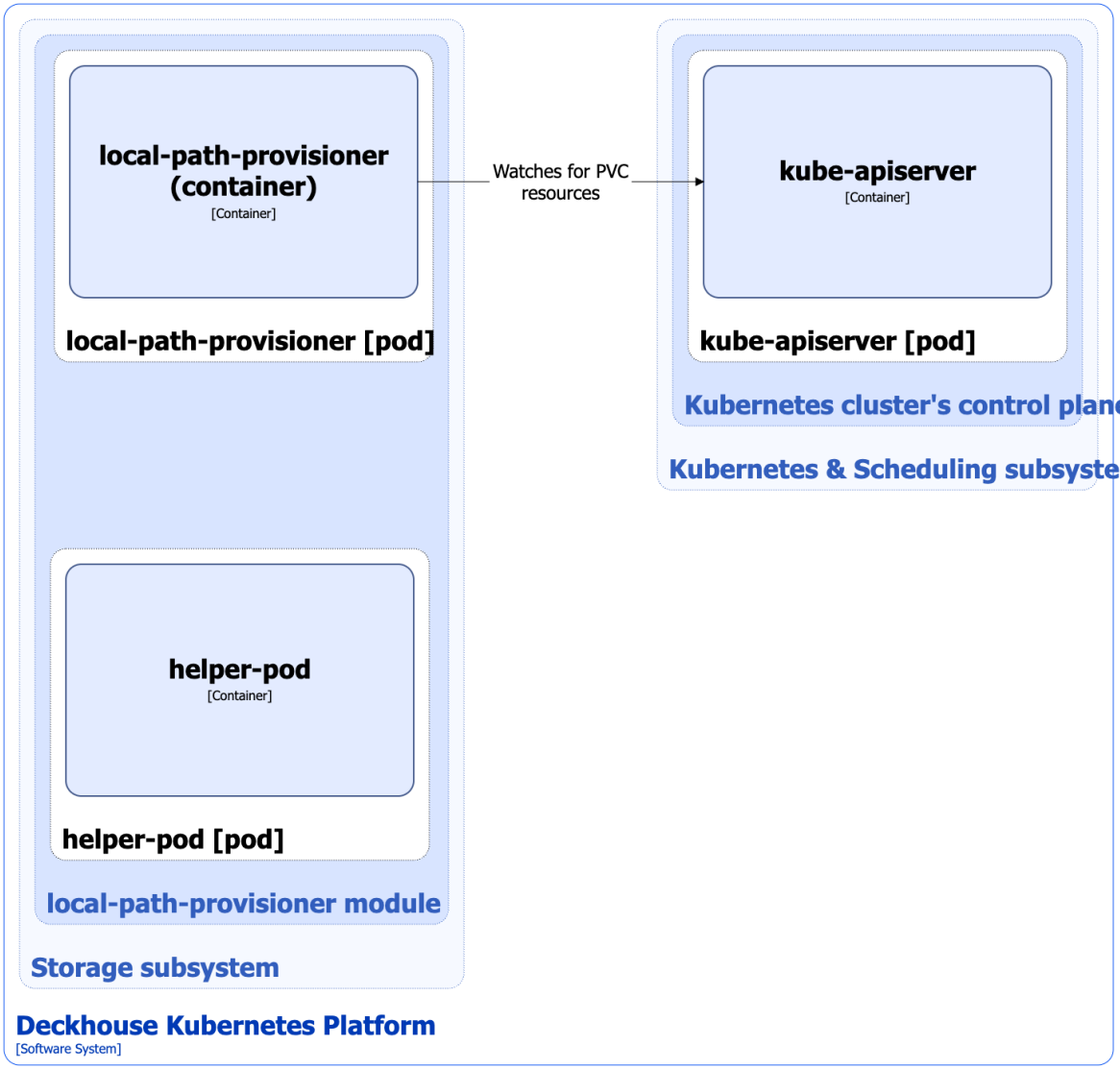
The `local-path-provisioner` module provides the local storage on Kubernetes nodes using `HostPath` volumes and creates StorageClass resources to manage the allocation of local storage.

For more details about module, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `local-path-provisioner` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagrams:



Module components

The module consists of the following components:

1. **Local-path-provisioner**: It performs the following actions when a pod orders a disk:
   - Creates a PersistentVolume of the `HostPath` volume type.
   - Creates a local disk directory for the volume on the target node. The directory path is based on the `path` parameter of the LocalPathProvisioner custom resource, as well as the PersistentVolume and PersistentVolumeClaim names.

   It consists of a single container:

   - **local-path-provisioner**: It is an open source project.
2. **Helper-pod**: It runs an installation script (`SETUP`) on the node before creating the volume to prepare the volume directory on the node and a cleanup script (`TEARDOWN`) after deleting the volume to clear the volume directory on the node.

   It consists of a single container:

   - **helper-pod**

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Manages PersistentVolumeClaim resources.

Additional resources

- Module `local-path-provisioner` documentation

Snapshot-controller module

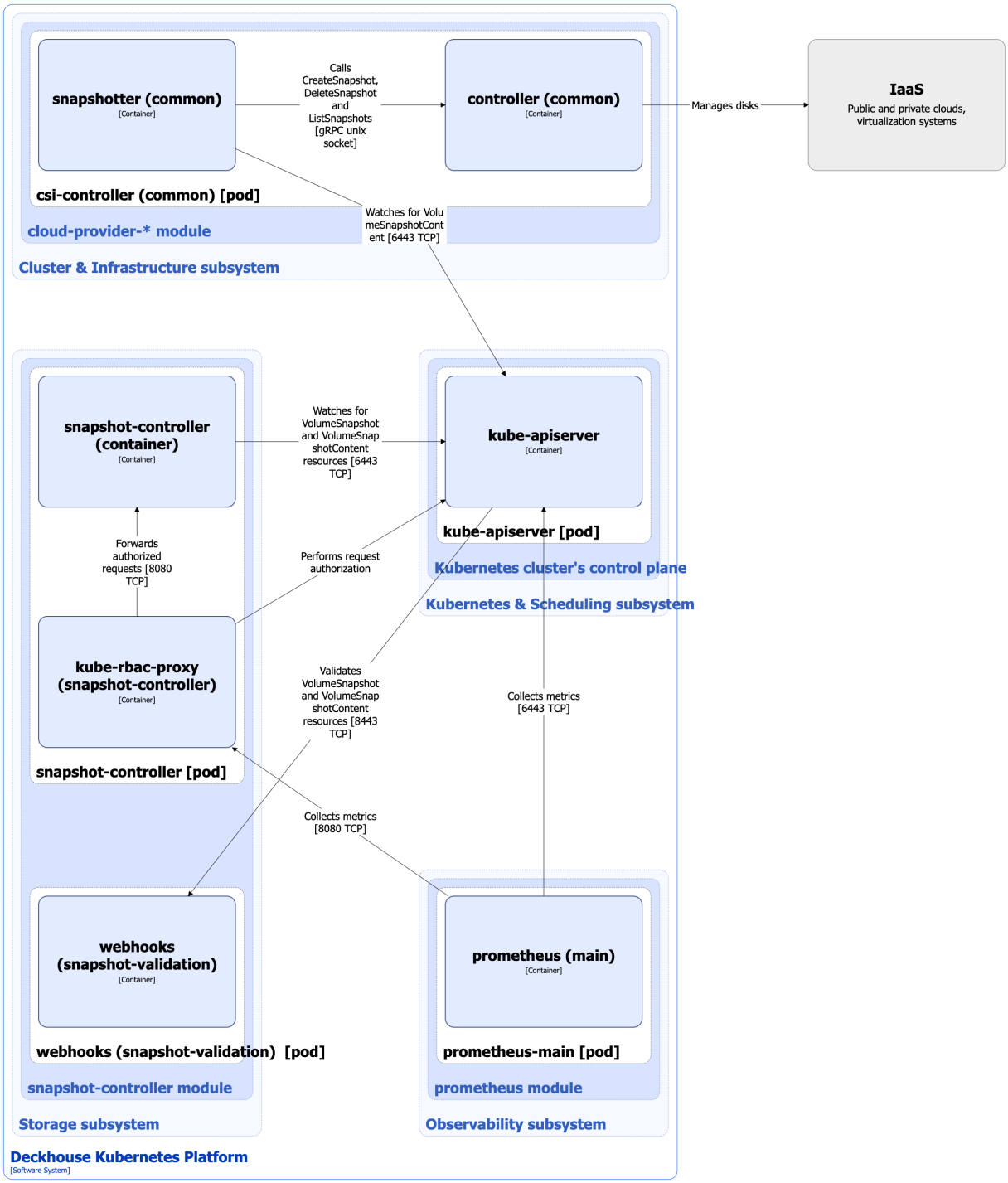
The `snapshot-controller` enables snapshot support for compatible CSI-drivers in Deckhouse Kubernetes Platform (DKP).

For more details about the module, refer to the corresponding documentation section.

Module architecture

> The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `snapshot-controller` module and its interactions with other components of DKP are shown in the following diagrams:



Module components

The module consists of the following components:

1. **Snapshot-controller**: Works in conjunction with the snapshotter (external-snapshotter) sidecar container of the csi-controller Pod of the `cloud-provider-*` module (as long as the provider's CSI driver supports snapshot creation).

   A single snapshot-controller is used for all installed CSI drivers. It watches for VolumeSnapshot and VolumeSnapshotContent resources. When a new VolumeSnapshot resource is created, the controller creates a VolumeSnapshotContent resource and binds them together. As a result, VolumeSnapshot points to a respective VolumeSnapshotContent, and that VolumeSnapshotContent points to the original VolumeSnapshot.

   Snapshot creation is multi-step process:

   1. Snapshot-controller creates a VolumeSnapshotContent resource.
   2. The snapshotter sidecar triggers snapshot creation through csi-controller on the corresponding node and updates the VolumeSnapshotContent status (`snapshotHandle`, `creationTime`, `restoreSize`, `readyToUse`, and `error` fields).
   3. Snapshot-controller watches for the VolumeSnapshotContent status and keeps updating the VolumeSnapshot status until the bi-directional binding is complete and the `readyToUse` field is set to `true`. If any error occurs, the `error` field is updated accordingly.

   It consists of the following containers:

   - **snapshot-controller**: It is an open-source project.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the controller metrics. It is an open-source project.

2. **Webhooks**: Component implementing a webhook server used for validating the VolumeSnapshot and VolumeSnapshotContent resources through Validating Admission Controllers. It consists of a single container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for VolumeSnapshot and VolumeSnapshotContent resources.
   - Authorizes requests for controller metrics.

The following external components interact with the module:

1. **Kube-apiserver**: Uses the validation webhook to verify the created VolumeSnapshot and VolumeSnapshotContent resources.
2. **Prometheus-main**: Collects metrics from the controller.

Additional resources

- Module `snapshot-controller` documentation

Sds-local-volume module

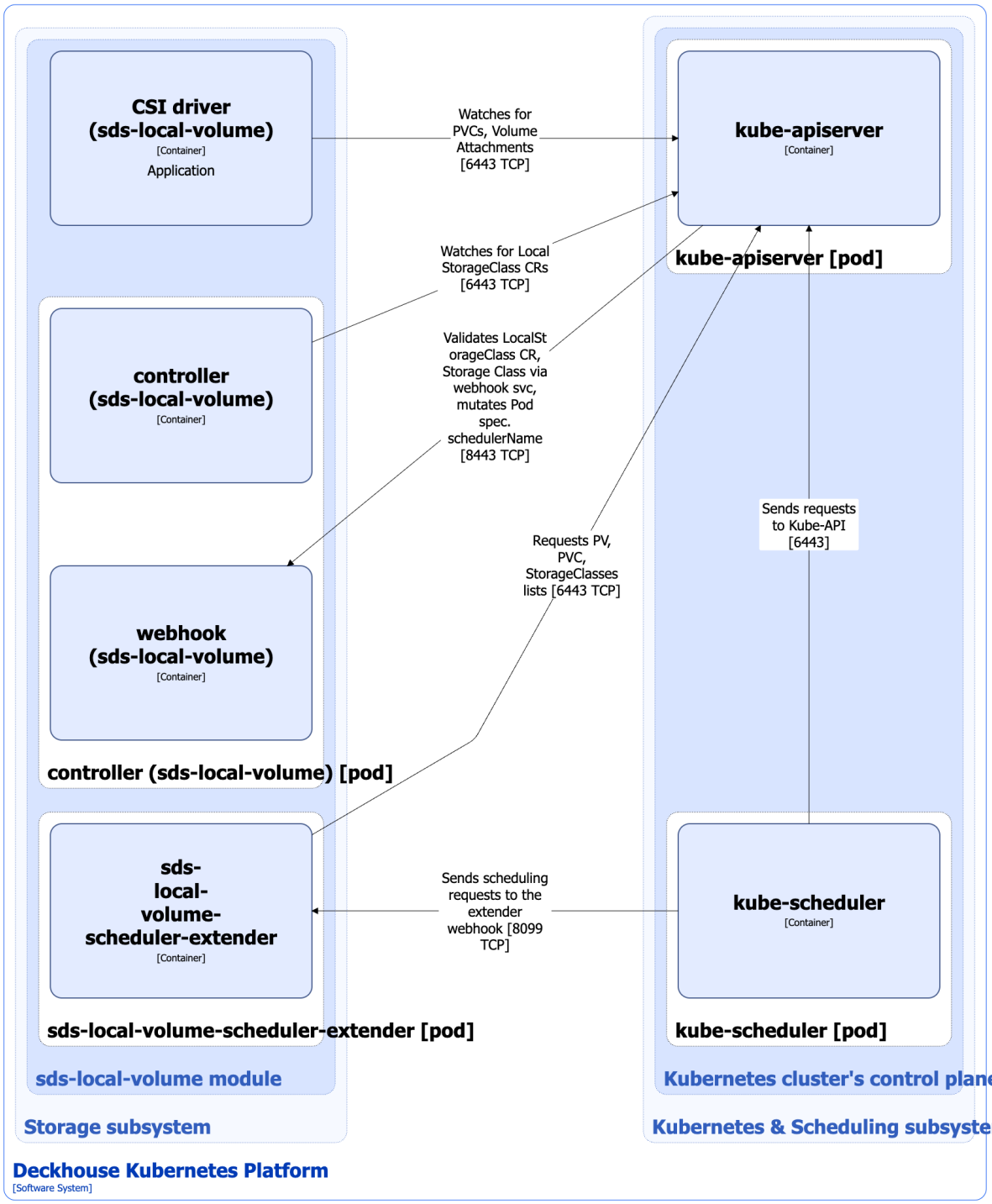
The `sds-local-volume` module is designed to manage local block storage based on LVM. It enables creating StorageClasses in Kubernetes using the LocalStorageClass resource.

For more details about module, refer to the module documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `sds-local-volume` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagrams:



Module components

The module consists of the following components:

1. **Controller**: It reconciles LocalStorageClass custom resources. LocalStorageClass is a Kubernetes custom resource that defines the configuration for Kubernetes StorageClass. The StorageClass being created uses `local.csi.storage.deckhouse.io` provisioner. StorageClass configures LVM logical volume types, VolumeGroups settings, reclaim policy, volume binding mode, etc. These settings are used by the provisioner of CSI driver (`sds-local-volume`) when managing LVM-based local volumes.

   It consists of the following containers:

   - **controller**: Main container.
   - **webhook**: A sidecar container that implements a webhook server for validating LocalStorageClass custom resources, StorageClass resources, as well as for mutating the `spec.schedulerName` attribute of pods using volumes created by `local.csi.storage.deckhouse.io` provisioner. As a result of the mutation, `sds-local-volume` is filled in the name of the scheduler (`spec.schedulerName`) in the pod specification, so that the placement of the pod is determined not by the standard Kubernetes scheduler (kube-scheduler), but by the `sds-local-volume-scheduler-extender` component from this module.

2. **Sds-local-volume-scheduler-extender**: It consists of a single container. It is a kube-scheduler extender, which implements a scheduling logic specific for pods using local volumes. When planning, the free space on the nodes used to place local volumes on them is taken into account, as well as the size of the disk space that needs to be

reserved for these volumes.

3. **CSI driver (`sds-local-volume`)**: It is an implementation of the CSI driver for `local.csi.storage.deckhouse.io`. To study the CSI driver typical architecture used in DKP, refer to the CSI-driver architecture documentation section. CSI driver (`sds-local-volume`) is developed by Flant.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolume, PersistentVolumeClaim, VolumeAttachment and StorageClass resources.
   - Reconciles LocalStorageClass custom resources.
   - Creates StorageClass resources.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Validates LocalStorageClass custom resources and StorageClass resources.
   - Mutates `spec.schedulerName` attribute for pods which use volumes created by `local.csi.storage.deckhouse.io` provisioner.
2. **Kube-scheduler**: Sends scheduling requests to the `sds-local-volume-scheduler-extender` webhook for the pods with `sds-local-volume` value specified in the `spec.schedulerName` attribute.

Additional resources

- Module `sds-local-volume` documentation

Sds-node-configurator module

The `sds-node-configurator` manages block devices and LVM on Kubernetes cluster nodes through Kubernetes custom resources, performing following operations:

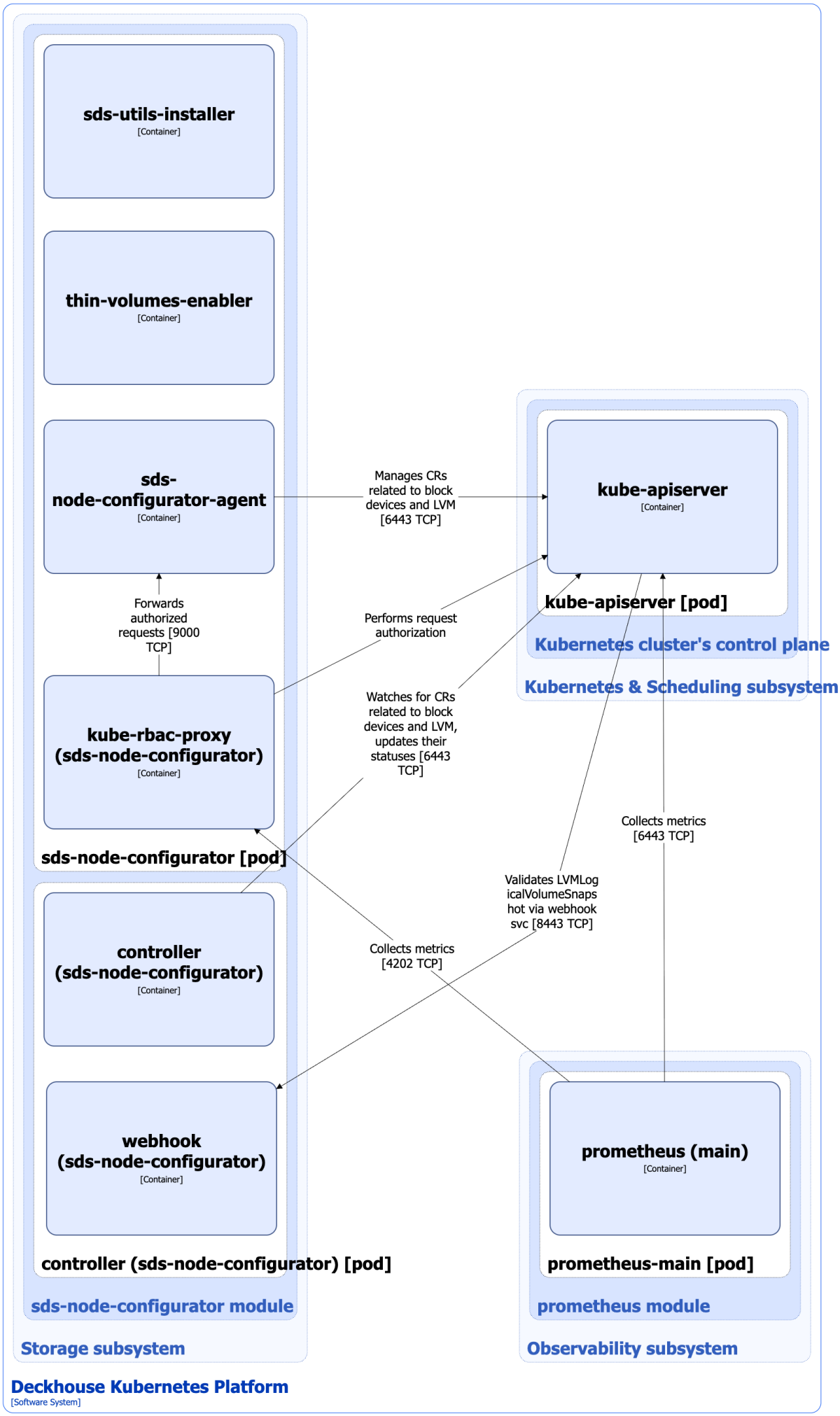
- Automatic discovery of block devices and creation/update/deletion of corresponding BlockDevice custom resources.
- Automatic discovery of LVM Volume Groups with `storage.deckhouse.io/enabled=true` LVM tag and thin pools on them on nodes, as well as management of corresponding LVMVolumeGroup resources. The module automatically creates a LVMVolumeGroup resource if it doesn't exist yet for the discovered LVM Volume Group.
- Scanning LVM Physical Volumes on nodes that are part of managed LVM Volume Groups. When underlying block devices are expanded, corresponding LVM Physical Volumes are automatically increased (performs pvresize).
- Creation/expansion/deletion of LVM Volume Groups on the node according to LVMVolumeGroup resource settings.

For more details about the module, refer to the module documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `sds-node-configurator` module and its interactions with other components of DKP are shown in the following diagrams:


sds-utils-installer
[Container]
thin-volumes-enabler
[Container]
sds-node-configurator-agent
[Container]
Manages CRs related to block devices and LVM [6443 TCP]
kube-apiserver
[Container]
kube-apiserver [pod]
Kubernetes cluster's control plane
Kubernetes & Scheduling subsystem
Forwards authorized requests [9000 TCP]
Performs request authorization
kube-rbac-proxy (sds-node-configurator)
[Container]
Watches for CRs related to block devices and LVM, updates their statuses [6443 TCP]
Collects metrics [6443 TCP]
sds-node-configurator [pod]
Validates LVMLogicalVolumeSnapshot via webhook svc [8443 TCP]
controller (sds-node-configurator)
[Container]
Collects metrics [4202 TCP]
webhook (sds-node-configurator)
[Container]
prometheus (main)
[Container]
controller (sds-node-configurator) [pod]
prometheus-main [pod]
sds-node-configurator module
prometheus module
Storage subsystem
Observability subsystem
Deckhouse Kubernetes Platform
[Software System]

Module components

The module consists of the following components:

1. **Sds-node-configurator** (DaemonSet): A controller running on cluster nodes and performing mentioned above operations with BlockDevice, LVMVolumeGroup, LVMLogicalVolume, LVMLogicalVolumeSnapshot custom resources, etc. The full list of resources managed by the module is provided in the module documentation.

   It consists of the following containers:

   - **sds-utils-installer**: Init container that installs a set of utilities necessary for managing block devices and LVM volumes.
   - **thin-volumes-enabler**: Init container that enables thin volumes.
   - **sds-node-configurator-agent**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the controller metrics. It is an open-source project.

2. **Controller** (Deployment): A controller that monitors custom resources related to block devices and LVM. The controller works with the metadata of resources and updates their statuses.

   It consists of the following containers:

   - **controller**: Main container.
   - **webhook**: Sidecar container that implements a webhook server for LVMLogicalVolumeSnapshot custom resources validation. If used edition of DKP does not support LVM logical volume snapshots functionality, LVMLogicalVolumeSnapshot custom resource does not pass validation.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Reconciles custom resources related to block devices and LVM.
   - Authorizes requests for metrics.

The following external components interact with the module:

1. **Kube-apiserver**: Validates LVMLogicalVolumeSnapshot custom resources.
2. **Prometheus-main**: Collects `sds-node-configurator` metrics.

Additional resources

- Module `sds-node-configurator` documentation

Csi-ceph module

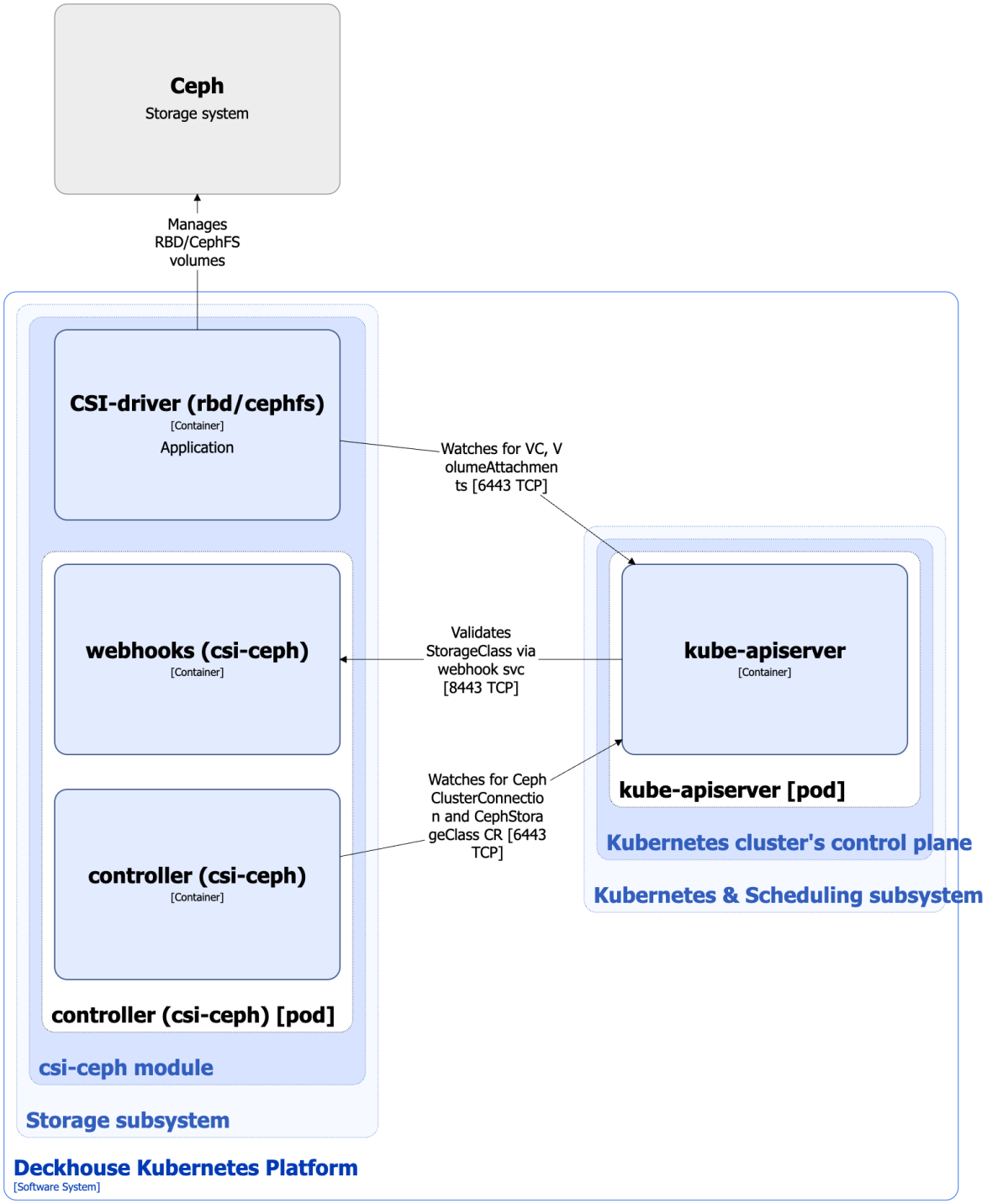
The `csi-ceph` module is designed to integrate DKP with Ceph clusters and provides storage management based on RBD (RADOS Block Device) or CephFS. It allows creating StorageClasses in Kubernetes using the CephStorageClass resource.

For more details about the module, refer to the module documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-ceph` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the following custom resources:

   - CephClusterConnection: Ceph cluster connection parameters.
   - CephStorageClass: Defines the configuration for the Kubernetes StorageClass that will be created.

   CephStorageClass specifies the storage class type (`CephFS` or `RBD`), reclaim policy, Ceph cluster connection parameters, and additional parameters specific to each storage class type. Depending on the storage class type, these parameters are used by the provisioner of either the `rbd.csi.ceph.com` or `cephfs.csi.ceph.com` CSI driver when managing volumes.

   The CephMetadataBackup custom resource is used in migration and recovery workflows implemented by module hooks, rather than by the main runtime controller.

   It consists of the following containers:

   - **controller**: Main container.
   - **webhooks**: A sidecar container that implements a webhook server for StorageClass resources validation.

2. **CSI driver (rbd/cephfs)**: An implementation of the CSI driver for the `rbd.csi.ceph.com` or `cephfs.csi.ceph.com` provisioner. The CSI driver is selected by setting the storage class type in the

CephStorageClass custom resource.

The `csi-cephfs` CSI driver follows the standard CSI driver architecture used in DKP.

The `csi-rbd` CSI driver uses an architecture that differs from the standard CSI driver architecture.

Module interactions

The module interacts with the following components:

- **Kube-apiserver**:
  - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
  - Reconciles CephClusterConnection and CephStorageClass custom resources.
  - Creates StorageClass resources.

The following external components interact with the module:

- **Kube-apiserver**: Validates StorageClass resources.

Additional resources

- Module `csi-ceph` documentation

Csi-hpe module

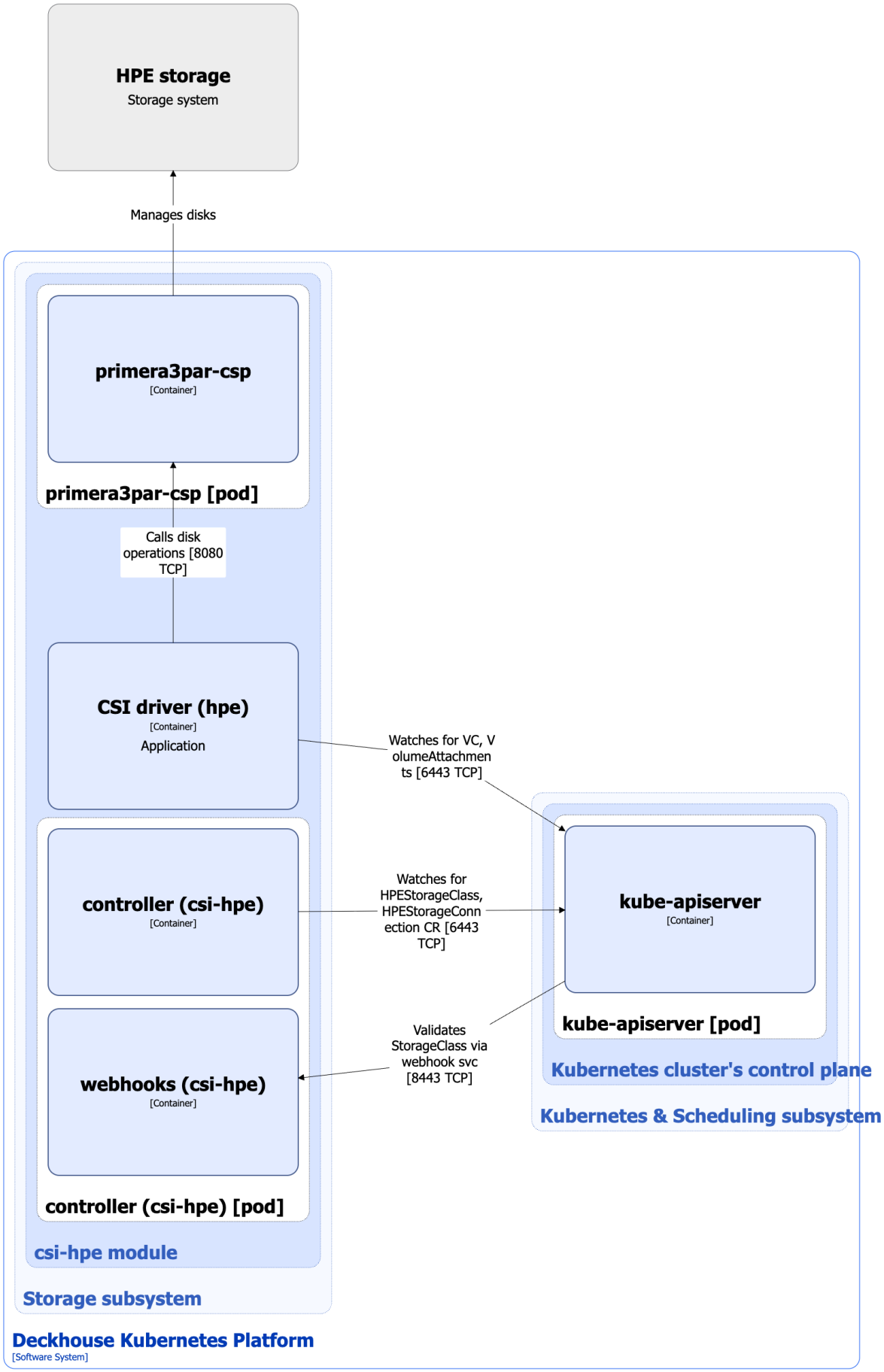
The `csi-hpe` module is designed to manage volumes using HPE storage systems. It enables creating StorageClass resources in Kubernetes using the HPEStorageClass custom resource.

For more details about the module, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-hpe` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the following custom resources:
   - HPEStorageConnection: Parameters for connecting to HPE storage systems.

- HPEStorageClass: Defines configuration for Kubernetes StorageClass.

  HPEStorageClass defines the connection protocol, resource pool name, filesystem type, and reclaim policy.

  The controller also synchronizes the `storage.deckhouse.io/csi-hpe-node` label on cluster nodes according to the `spec.settings.nodeSelector` value in the ModuleConfig custom resource.

  It consists of the following containers:

  - **controller**: Main container.
  - **webhooks**: Sidecar container implementing a webhook server for StorageClass validation.

2. **CSI driver (hpe)**: CSI driver implementation for the `csi.hpe.com` provisioner. To study the typical CSI driver architecture used in DKP, refer to the CSI driver documentation page.
3. **Primera3par-csp**: A service container provider (Container Storage Provider, CSP) responsible for integrating the CSI driver with HPE Primera and 3PAR storage systems. It handles communication between Kubernetes and the storage arrays, manages session handling, and path replication.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
   - Reconciles HPEStorageConnection and HPEStorageClass custom resources.
   - Creates and updates VolumeSnapshotClass, Secret, and StorageClass resources.
2. **HPE storage system**: Creates, deletes, and manages volumes, and provides multipath data access.

The following external components interact with the module:

1. **Kube-apiserver**: Validates StorageClass resources.

Additional resources

- Module `csi-hpe` documentation

Csi-huawei module

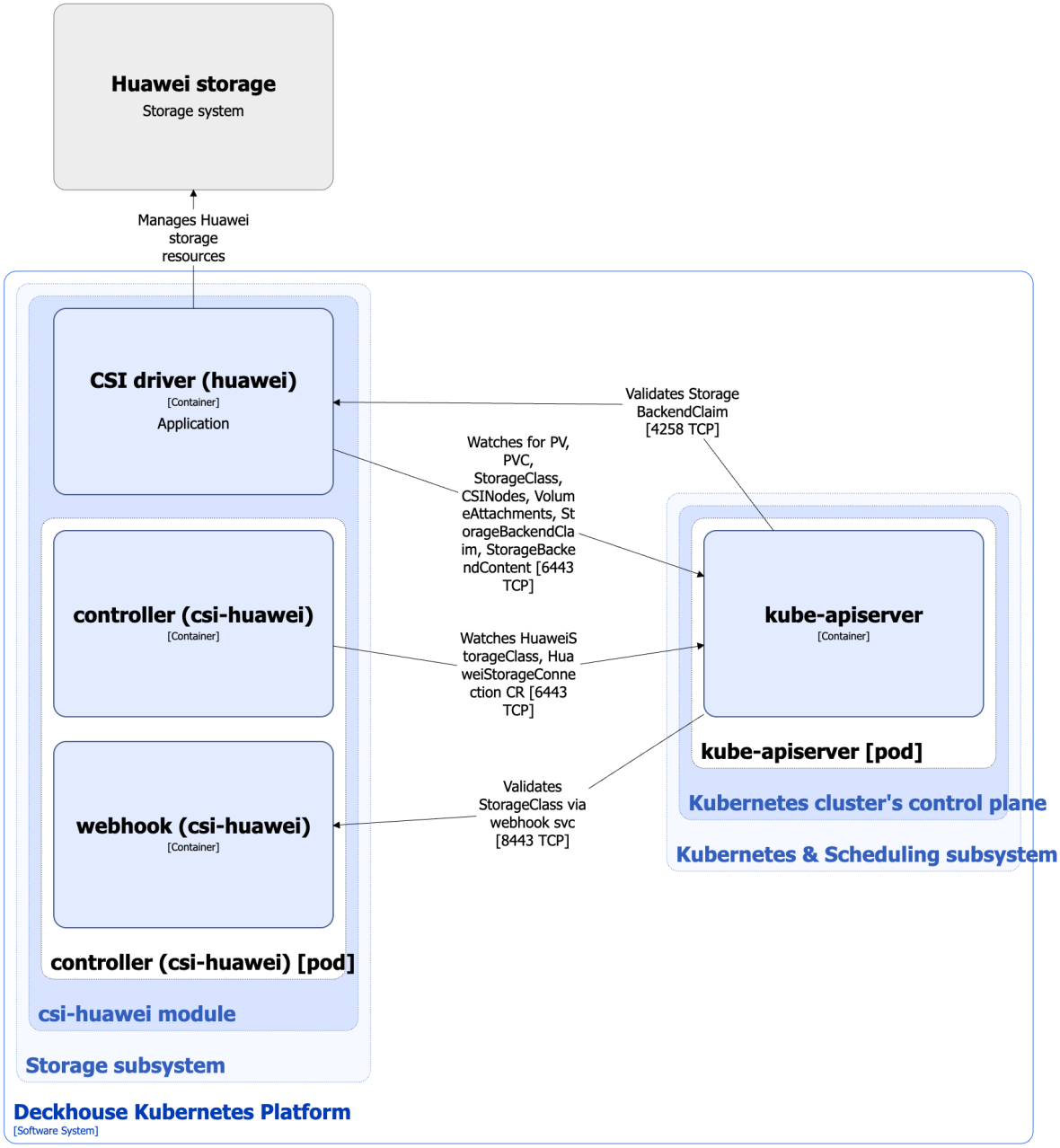
The `csi-huawei` module is designed to manage volumes using Huawei storage systems. It enables creating StorageClass resources in Kubernetes using the HuaweiStorageClass custom resource.

For more details about the module, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-huawei` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the following custom resources:
   - HuaweiStorageConnection: Parameters for connecting to Huawei storage systems.
   - HuaweiStorageClass: Defines configuration for Kubernetes StorageClass.

   HuaweiStorageClass defines the resource pool name, filesystem type, and reclaim policy.

   It consists of the following containers:

   - **controller**: Main container.
   - **webhooks**: Sidecar container implementing a webhook server for StorageClass validation.
2. **CSI driver (huawei)**: CSI driver implementation for the `csi.huawei.com` provisioner. To study the architecture of the `csi-huawei` CSI driver, refer to the corresponding documentation page.

   The `csi-huawei` CSI driver reconciles the following custom resources:

   - StorageBackendClaim: A request to connect to Huawei storage.
   - StorageBackendContent: Description of the actual connection to Huawei storage.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

- Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
- Reconciles HuaweiStorageConnection, HuaweiStorageClass, StorageBackendClaim, and StorageBackendContent custom resources.
- Creates StorageClass resources.

2. **Huawei storage system**: Creates and deletes volumes, and attaches/detaches volumes to/from nodes.

The following external components interact with the module:

1. **Kube-apiserver**: Validates StorageBackendClaim custom resources and standard StorageClass resources.

Additional resources

- Module `csi-huawei` documentation

Csi-netapp module

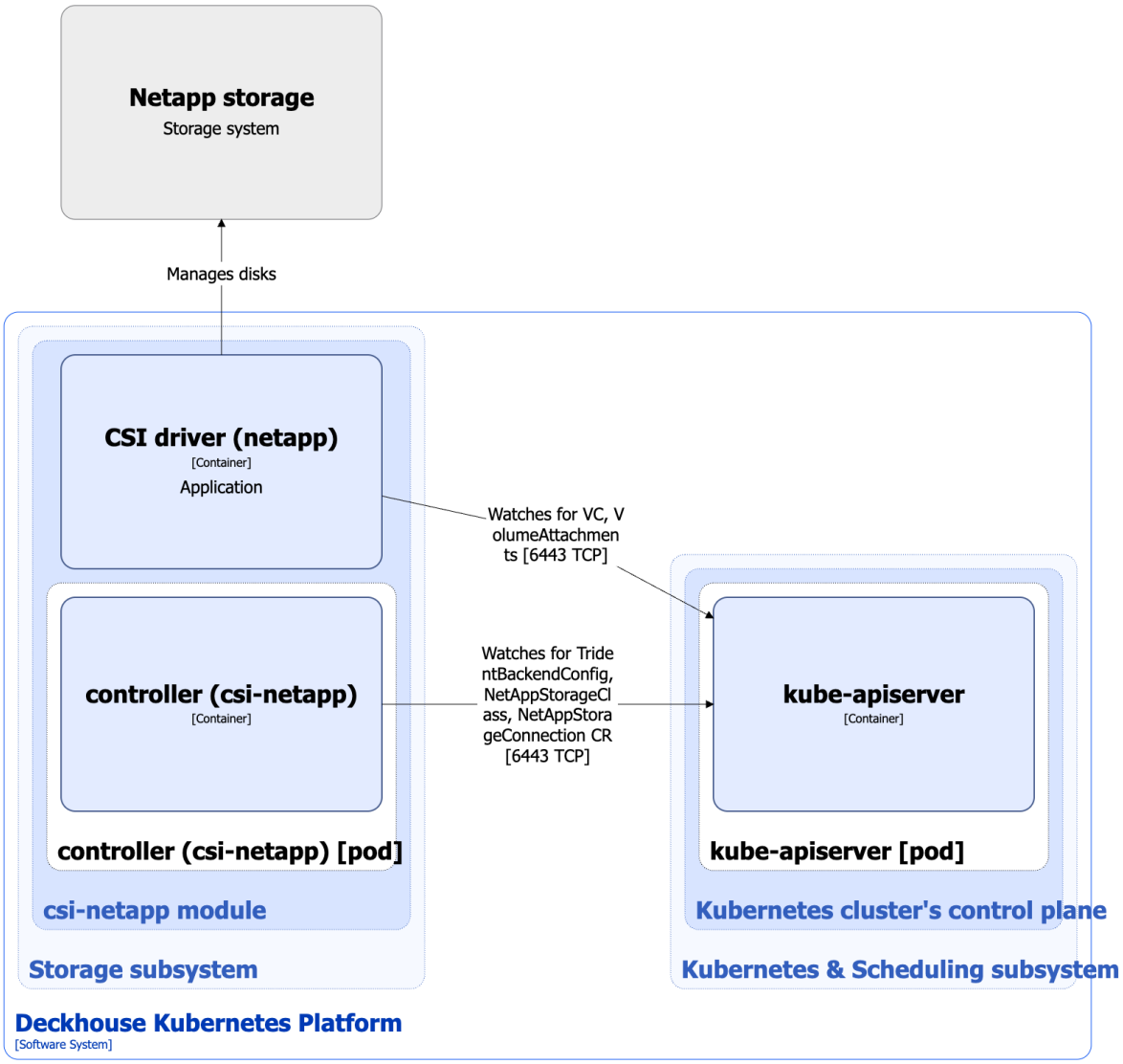
The `csi-netapp` module is designed to manage volumes using NetApp storage systems. It enables creating StorageClass resources in Kubernetes using the NetappStorageClass custom resource.

For more details about the module, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-netapp` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the following custom resources:
   - NetappStorageConnection: Parameters for connecting to NetApp storage systems.
   - NetappStorageClass: Defines configuration for creating Kubernetes StorageClass that uses the `csi.trident.netapp.io` provisioner.

   The controller also synchronizes the `storage.deckhouse.io/csi-netapp-node` label on cluster nodes according to the `spec.settings.nodeSelector` value in the ModuleConfig custom resource.

   It consists of a single container, **controller**.
2. **CSI driver (netapp)**: CSI driver implementation for the `csi.trident.netapp.io` provisioner. To study the typical CSI driver architecture used in DKP, refer to the CSI driver documentation page.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
   - Reconciles TridentBackendConfig, NetappStorageConnection, and NetappStorageClass custom resources.
   - Creates VolumeSnapshotClass, Secret, and StorageClass resources.
2. **NetApp storage system**: Creates, deletes, and manages volumes, and attaches/detaches volumes to/from nodes.

Additional resources

- Module `csi-netapp` documentation

Csi-nfs module

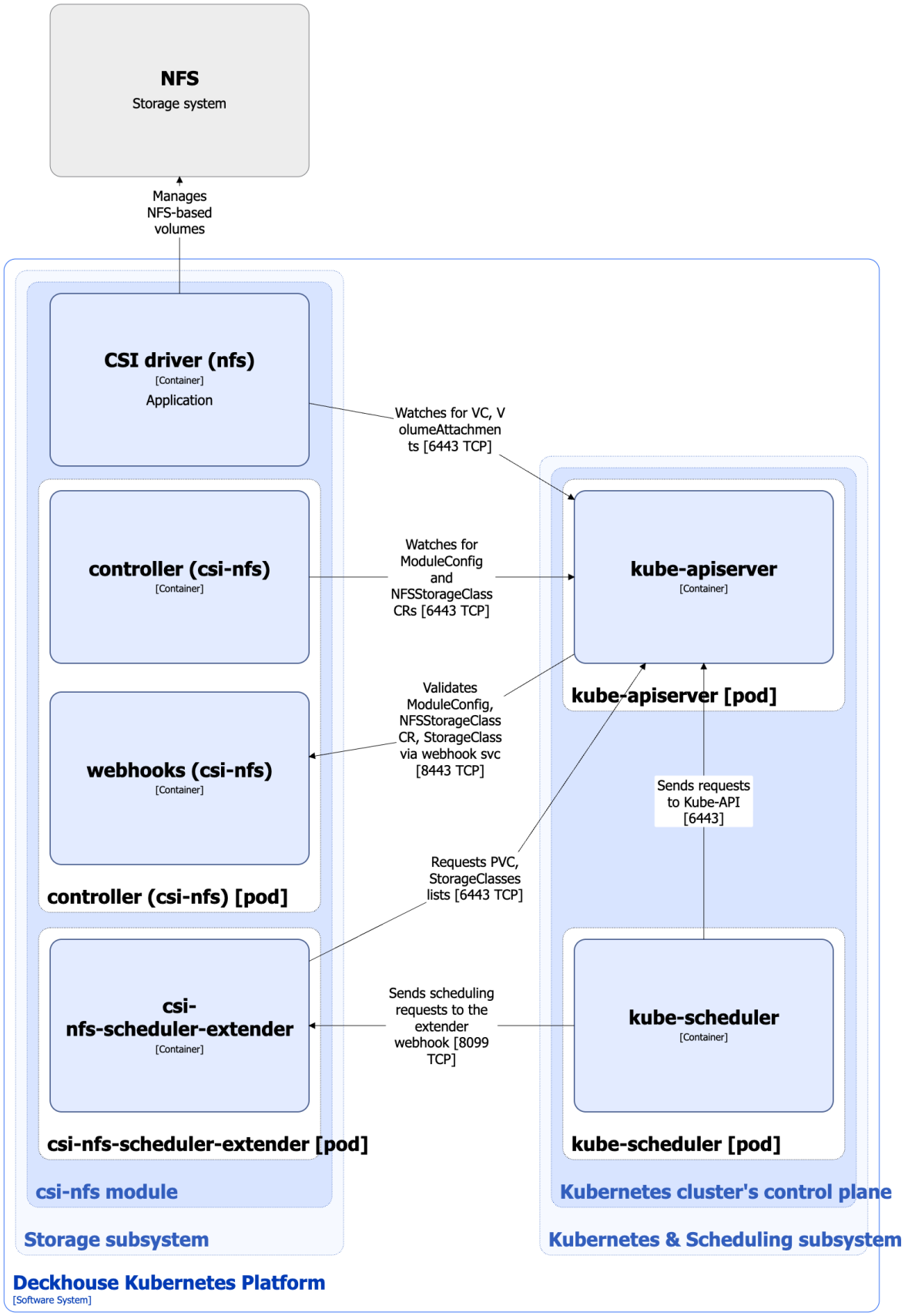
The `csi-nfs` module is designed to manage NFS-based volumes. It enables creating StorageClasses in Kubernetes using the NFSStorageClass resource.

For more details about the module, refer to the module documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-nfs` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagrams:



Module components

The module consists of the following components:

1. **Controller**: It reconciles NFSStorageClass custom resources. NFSStorageClass defines the configuration for the created Kubernetes StorageClass, which uses the `nfs.csi.k8s.io` provisioner.

   The created StorageClass configures the NFS server connection settings, a reclaim policy, the volume binding mode, etc. These settings are used by the provisioner of the `csi-nfs` CSI driver when managing NFS-based volumes.

   Also the controller sets the `storage.deckhouse.io/csi-nfs-node` label on the cluster nodes according to the

`spec.workloadNodes.nodeSelector` parameter value of the NFSStorageClass custom resource.

It consists of the following containers:

- **controller**: Main container.
- **webhooks**: A sidecar container that implements a webhook server for validating ModuleConfig and NFSStorageClass custom resources, as well as StorageClass resources.

2. **Csi-nfs-scheduler-extender**: It consists of a single container. It is a kube-scheduler extender, which implements a scheduling logic specific for pods using NFS-based volumes. When scheduling pods, it relies on the node selectors set in the NFSStorageClass custom resource.

   The **csi-nfs-scheduler-extender** may be absent if node selectors are not set in the NFSStorageClass custom resource.

3. **CSI driver (`csi-nfs`)**: It is an implementation of the CSI driver for `nfs.csi.k8s.io` (NFS CSI driver). To study the `csi-nfs` CSI driver architecture, refer to the CSI driver documentation page.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches for PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
   - Reconciles NFSStorageClass custom resources.
   - Creates StorageClass resources.

The following external components interact with the module:

1. **Kube-apiserver**: Validates ModuleConfig and NFSStorageClass custom resources, as well as StorageClass resources.
2. **Kube-scheduler**: Sends scheduling requests to the `csi-nfs-scheduler-extender` webhook for the pods using NFS-based volumes.

Additional resources

- Module `csi-nfs` documentation

Csi-s3 module

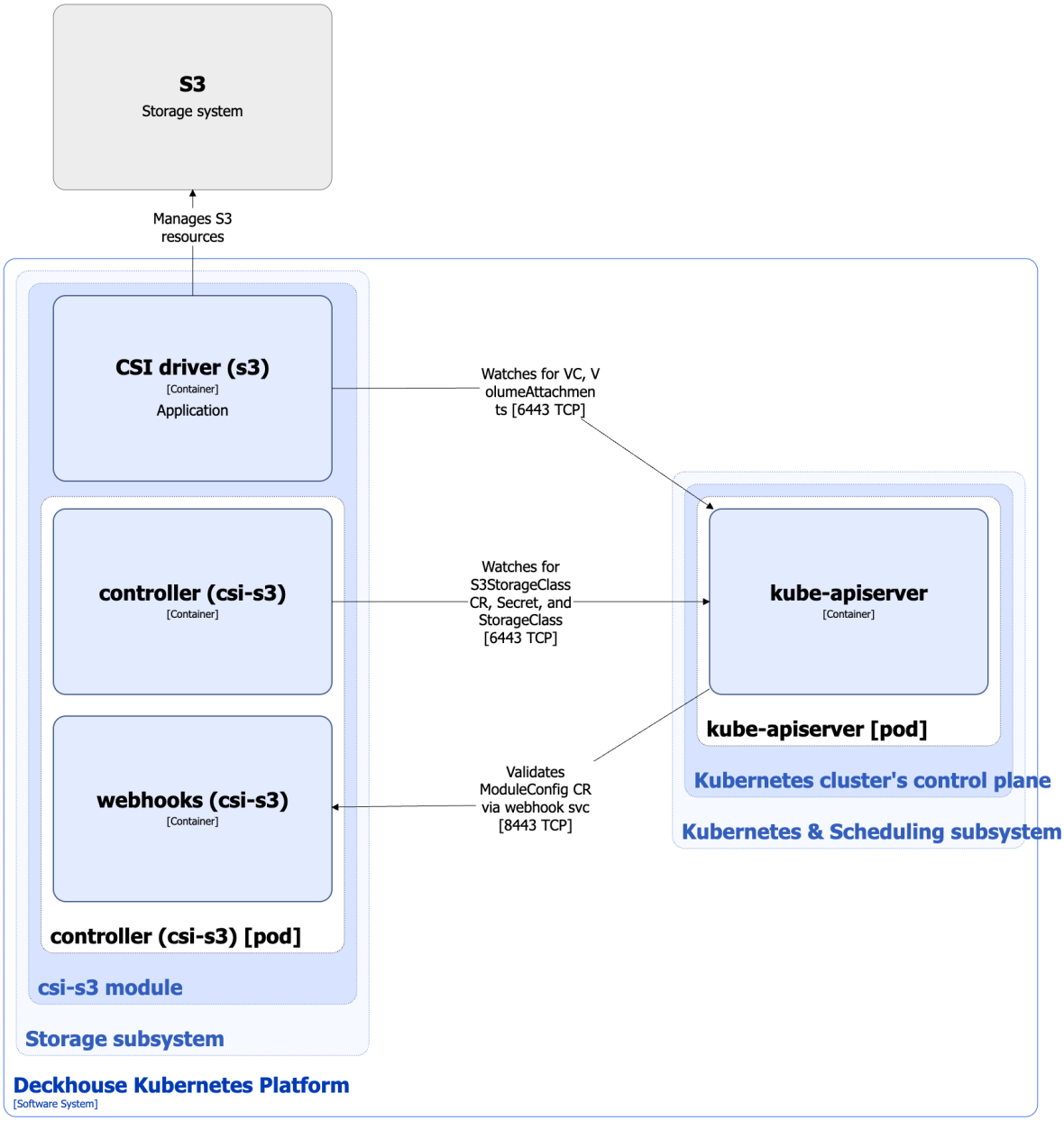
The `csi-s3` module is designed to manage volumes based on S3 storage. The module uses geeseFS, an S3-backed FUSE filesystem. The module enables creating StorageClass resources in Kubernetes using the S3StorageClass custom resource.

For more details about the module, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-s3` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the S3StorageClass custom resource. The S3StorageClass resource defines configuration for a Kubernetes StorageClass that uses the `ru.yandex.s3.csi` provisioner.

   It consists of the following containers:

   - **controller**: Main container.
   - **webhooks**: Sidecar container implementing a webhook server for ModuleConfig custom resource validation.

2. **CSI driver (s3)**: CSI driver implementation for the `ru.yandex.s3.csi` provisioner. To study the architecture of the `csi-s3` CSI driver, refer to the CSI driver documentation page.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, Secret, and StorageClass resources.
   - Reconciles S3StorageClass and ModuleConfig custom resources.
   - Creates StorageClass and Secret resources.

2. **S3 storage**: Creates and deletes volumes, and attaches/detaches volumes to/from nodes.

The following external components interact with the module:

1. **Kube-apiserver**: Validates ModuleConfig custom resources.

Additional resources

- Module `csi-s3` documentation

Csi-scsi-generic module

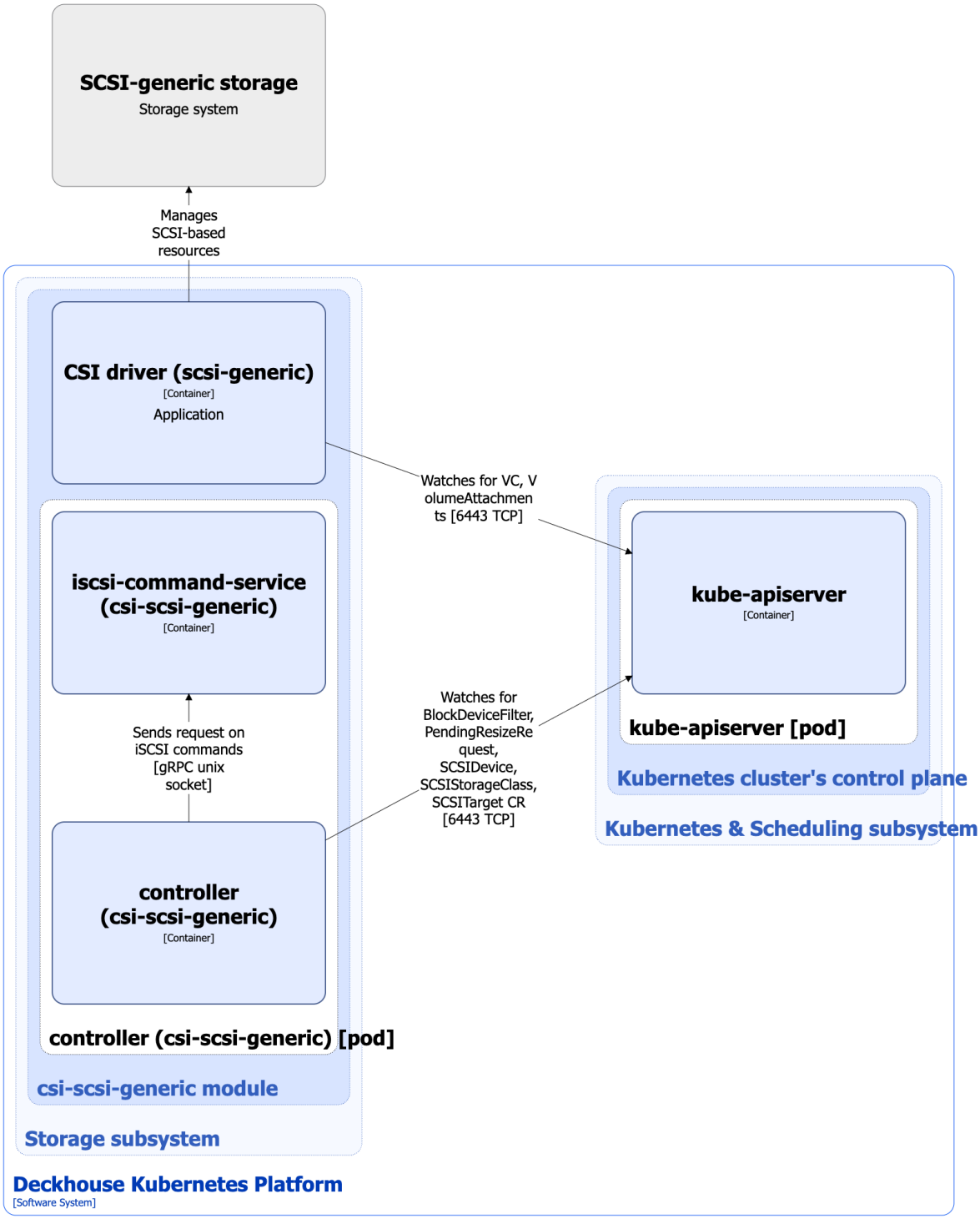
The `csi-scsi-generic` module is designed to manage volumes on storage systems connected over SCSI. It enables creating StorageClass resources in Kubernetes using the SCSIStorageClass resource and supports both iSCSI and Fibre Channel (FC) technologies.

For more details about the module, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-scsi-generic` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the following custom resources:
   - SCSITarget: Describes a storage connection endpoint (iSCSI/FC).
   - SCSIDevice: Describes a discovered SCSI device.
   - PendingResizeRequest: Represents a request for deferred PVC expansion when the requested size is larger than the current device size.
   - SCSIStorageClass: Defines configuration for Kubernetes StorageClass.
   - BlockDeviceFilter: Specifies filters for selecting physical SCSI devices that can be used to create volumes in the system. Provides flexible management of the available device pool by setting rules based on device attributes and including/excluding specific devices.

   SCSIStorageClass defines the device selector (`scsiDeviceSelector`), reclaim policy, and volume cleanup parameters.

   It consists of the following containers:

   - **controller**: Main container.
   - **iscsi-command-service**: Sidecar container implementing SCSI device discovery.

2. **CSI driver (`csi-scsi-generic`)**: CSI driver implementation for the `scsi-generic.csi.storage.deckhouse.io` provisioner. To study the architecture of the `csi-scsi-generic` CSI driver, refer to the CSI driver documentation page.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
   - Reconciles BlockDeviceFilter, SCSITarget, SCSIDevice, PendingResizeRequest, and SCSIStorageClass custom resources.
   - Creates StorageClass resources.
2. **SCSI-connected storage systems**: Coordinates the usage of available SCSI devices, including their binding and cleanup, as well as node attachment.

Additional resources

- Module `csi-scsi-generic` documentation

Csi-vsphere module

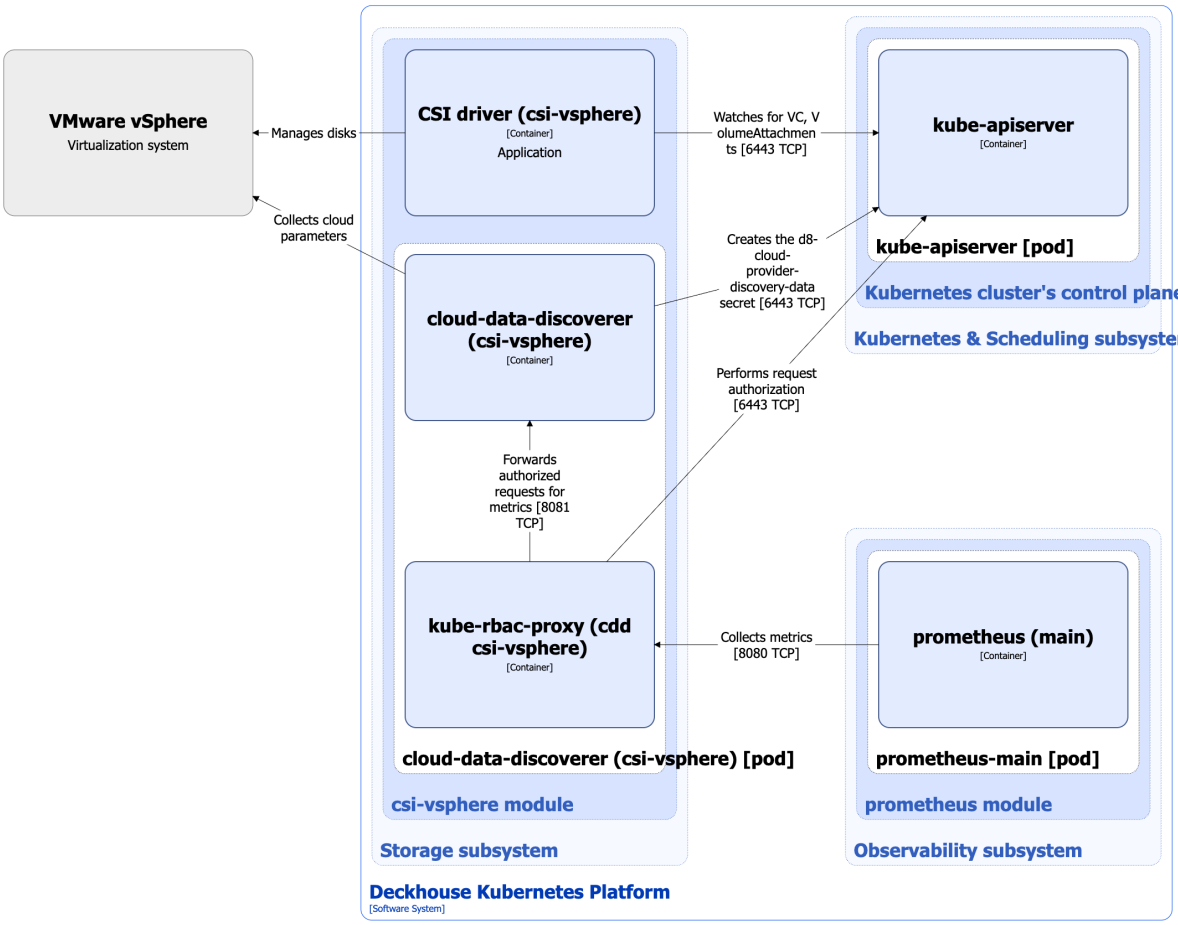
The `csi-vsphere` module provides Container Storage Interface (CSI) support for VMware vSphere environments, enabling dynamic provisioning and management of persistent storage volumes in Kubernetes clusters running on vSphere infrastructure.

For more details about the module configuration, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-vsphere` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Cloud-data-discoverer**: It is responsible for collecting data from the VMware vSphere API and providing it as a `kube-system/d8-cloud-provider-discovery-data` Secret. This secret contains the parameters of a specific cloud used CSI driver for volume management.

   It consists of the following containers:

   - **cloud-data-discoverer**: Main container.
   - **kube-rbac-proxy**: Sidecar container providing an RBAC-based authorization proxy for secure access to the cloud-data-discoverer metrics.

2. **CSI driver (vsphere)**: It is an implementation of the CSI driver for VMware vSphere. To study the `csi-vsphere` CSI driver architecture, refer to the corresponding documentation section.

   CSI driver (vsphere) does not support snapshots. For this reason, the `csi-controller` Pod does not include the snapshotter (external-snapshotter) sidecar container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:

   - Watches for PersistentVolumeClaim and VolumeAttachment custom resources.
   - Creates the `kube-system/d8-cloud-provider-discovery-data` Secret.
   - Creates StorageClass resources for each discovered datastore.
   - Authorizes the requests for metrics.

2. **VMware vSphere**:

   - Collects cloud parameters.
   - Manages disks.

The following external components interact with the module:

- **Prometheus-main**: Collects cloud-data-discoverer metrics.

Additional resources

- Module `csi-vsphere` documentation

Csi-yadro-tatlin-unified module

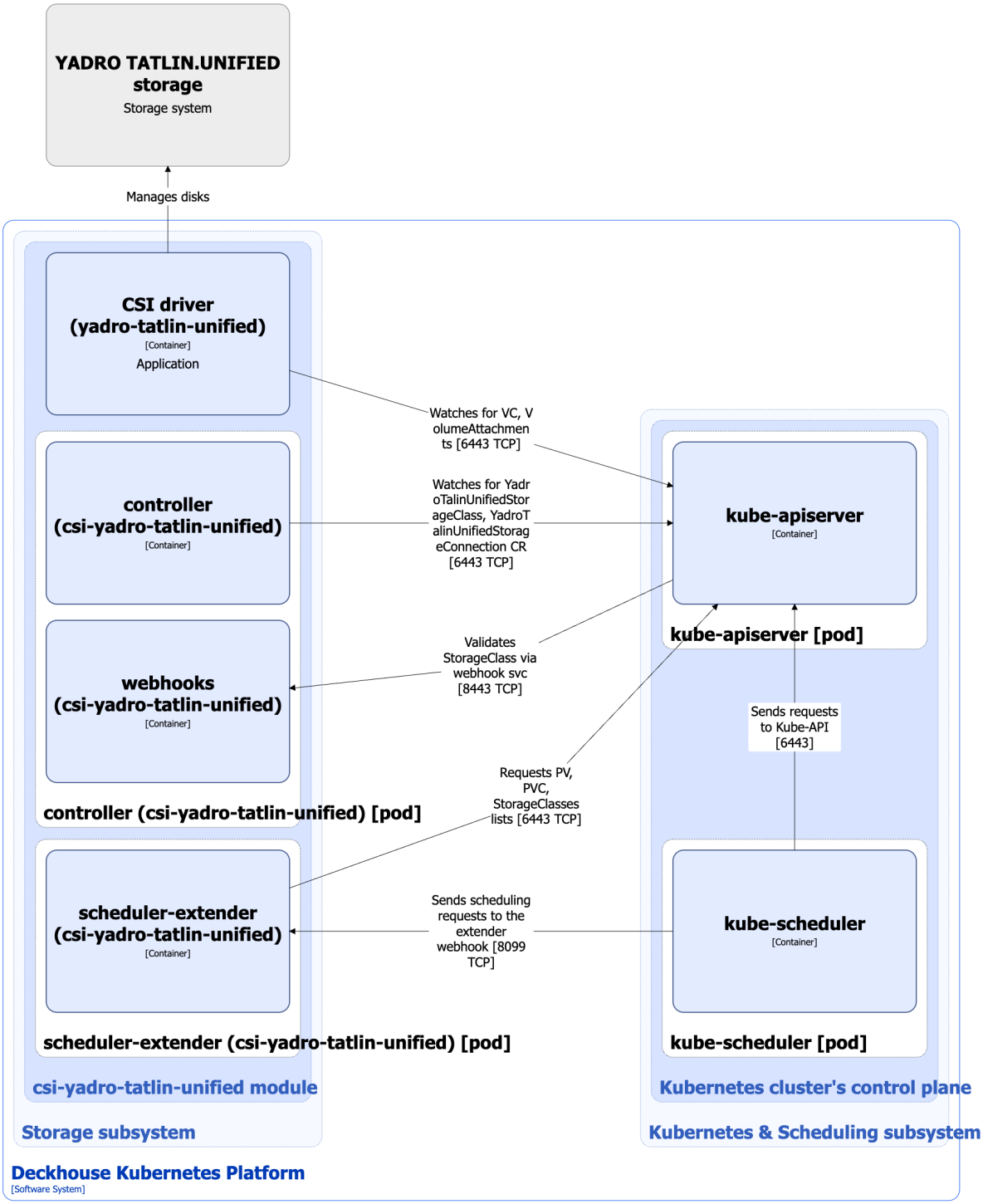
The `csi-yadro-tatlin-unified` module is designed to manage volumes using TATLIN.UNIFIED storage systems. It enables creating StorageClass resources in Kubernetes using the YadroTatlinUnifiedStorageClass custom resource.

For more details about the module, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-yadro-tatlin-unified` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Module components

The module consists of the following components:

1. **Controller**: A controller that reconciles the following custom resources:
   - YadroTatlinUnifiedStorageConnection: Parameters for connecting to TATLIN.UNIFIED storage systems.
   - YadroTatlinUnifiedStorageClass: Defines configuration for creating Kubernetes StorageClass that uses the `csi-tatlinunified.yadro.com` provisioner.

   YadroTatlinUnifiedStorageClass defines connection parameters, as well as resource pool name, filesystem type, and reclaim policy.

   It consists of the following containers:

   - **controller**: Main container.
   - **webhook**: Sidecar container implementing a webhook server for StorageClass resource validation.
2. **CSI driver (yadro-tatlin-unified)**: CSI driver implementation for the `csi-tatlinunified.yadro.com` provisioner. To study the typical CSI driver architecture used in DKP, refer to the CSI driver documentation page.
3. **Scheduler-extender**: A single-container component that acts as an extender for kube-scheduler. It implements scheduling logic specific to pods using volumes from TATLIN.UNIFIED storage systems. During scheduling, node selectors defined in the YadroTatlinUnifiedStorageConnection custom resource under the `controlPlane` and `dataPlane` parameters are taken into account.

   The component may be absent if node selectors are not set in the YadroTatlinUnifiedStorageConnection custom resource.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, and StorageClass resources.
   - Reconciles YadroTatlinUnifiedStorageConnection and YadroTatlinUnifiedStorageClass custom resources.
   - Creates StorageClass resources.
2. **YADRO TATLIN.UNIFIED storage system**: Creates and deletes volumes, and attaches/detaches volumes to/from nodes.

The following external components interact with the module:

1. **Kube-apiserver**: Validates standard StorageClass resources.

Additional resources

- Module `csi-yadro-tatlin-unified` documentation

Standard CSI driver

A CSI driver (plugin) is used to manage persistent storage volumes in Deckhouse Kubernetes Platform (DKP).

Container Storage Interface (CSI) is a standard interface that unifies access to storage systems and simplifies integration of different storage systems into clusters.

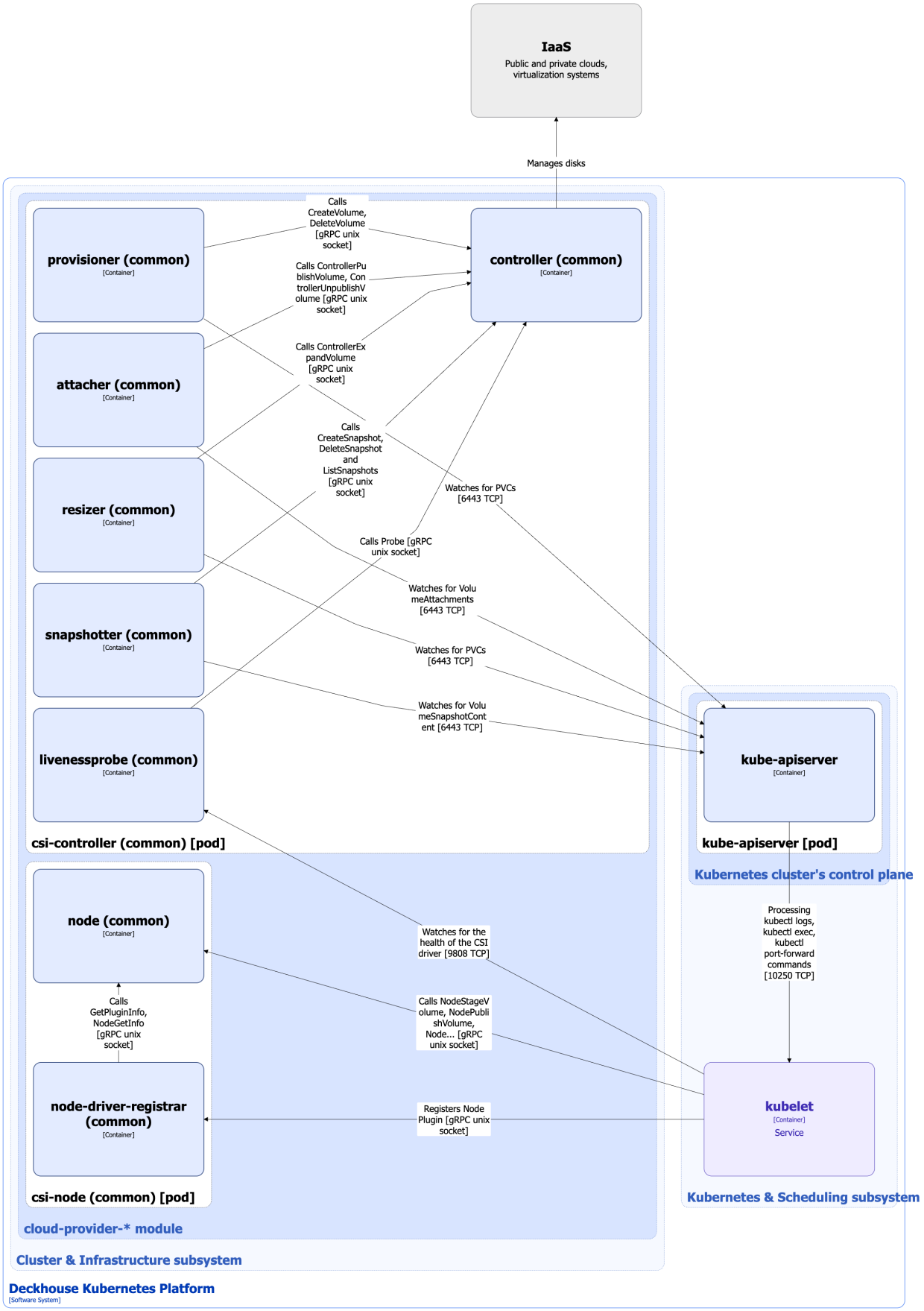
The CSI driver is included in DKP modules named `cloud-provider-*` and `csi-*`. Although each supported cloud provider or storage system uses its own implementation of the CSI specification, the architecture of the CSI driver is the same across all implementations. Implementations may differ in the set of supported capabilities and components.

The following is a description of the reference CSI driver architecture used in DKP. It includes all possible components and implements the full functionality used by DKP modules.

Driver architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The reference CSI driver architecture at Level 2 of the C4 model and its interactions with other DKP components are shown in the following diagram:



Driver components

The CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, attaching and detaching volumes from nodes, and managing snapshots. For example, in AWS this component calls the EC2 API to create EBS volumes.

   It consists of the following containers:

   - **controller**: Main container implementing CSI driver functionality (capabilities) through the gRPC services

Identity Service and Controller Service according to the CSI specification.

- **controller sidecar containers**: Kubernetes community-maintained external controllers.

  These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DKP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI driver functions in the controller container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

  External controllers communicate with the controller container over gRPC via Unix sockets.

  Csi-controller includes the following external controllers:

  - **provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`. It also uses `ValidateVolumeCapabilities` to verify compatibility.
  - **attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node and attaches or detaches volumes using the RPC methods `ControllerPublishVolume` and `ControllerUnpublishVolume`.
  - **resizer** (external-resizer): Watches updates to PersistentVolumeClaim resources and expands volumes using the `ControllerExpandVolume` RPC method when a user requests additional storage for a PVC, and the driver supports the `EXPAND_VOLUME` capability.
  - **snapshotter** (external-snapshotter): Works together with the `snapshot-controller` module, watches VolumeSnapshotContent resources, and manages volume snapshots using the RPC methods `CreateSnapshot`, `DeleteSnapshot`, and `ListSnapshots` (if supported by the driver).
  - **livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.

2. **Csi-node** (DaemonSet): Node Plugin running on all cluster nodes and responsible for local volume mount and unmount operations.

   ⚠ The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

   - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
   - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the node container occurs over gRPC via a Unix socket.

i Some sidecar containers from the external controller list (for example, snapshotter) may be absent in the Deployment of specific `cloud-provider-*` or `csi-*` modules if the corresponding functionality is not supported by the CSI driver implementation.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**: Monitors PersistentVolumeClaim, VolumeAttachment, and VolumeSnapshotContent resources.
2. **Cloud infrastructure** (or a virtualization system): Creates and deletes volumes, attaches and detaches volumes from nodes, and manages snapshots.

The following external components interact with the driver:

1. Kubelet:
   - Checks the CSI driver livenessProbe.
   - Registers the Node Plugin.
   - Calls the RPC methods `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, `NodeUnpublishVolume`, and `NodeExpandVolume` in Node Plugin.

   Kubelet communicates with Node Plugin via gRPC over a Unix socket.

Additional resources

- Module `snapshot-controller` documentation

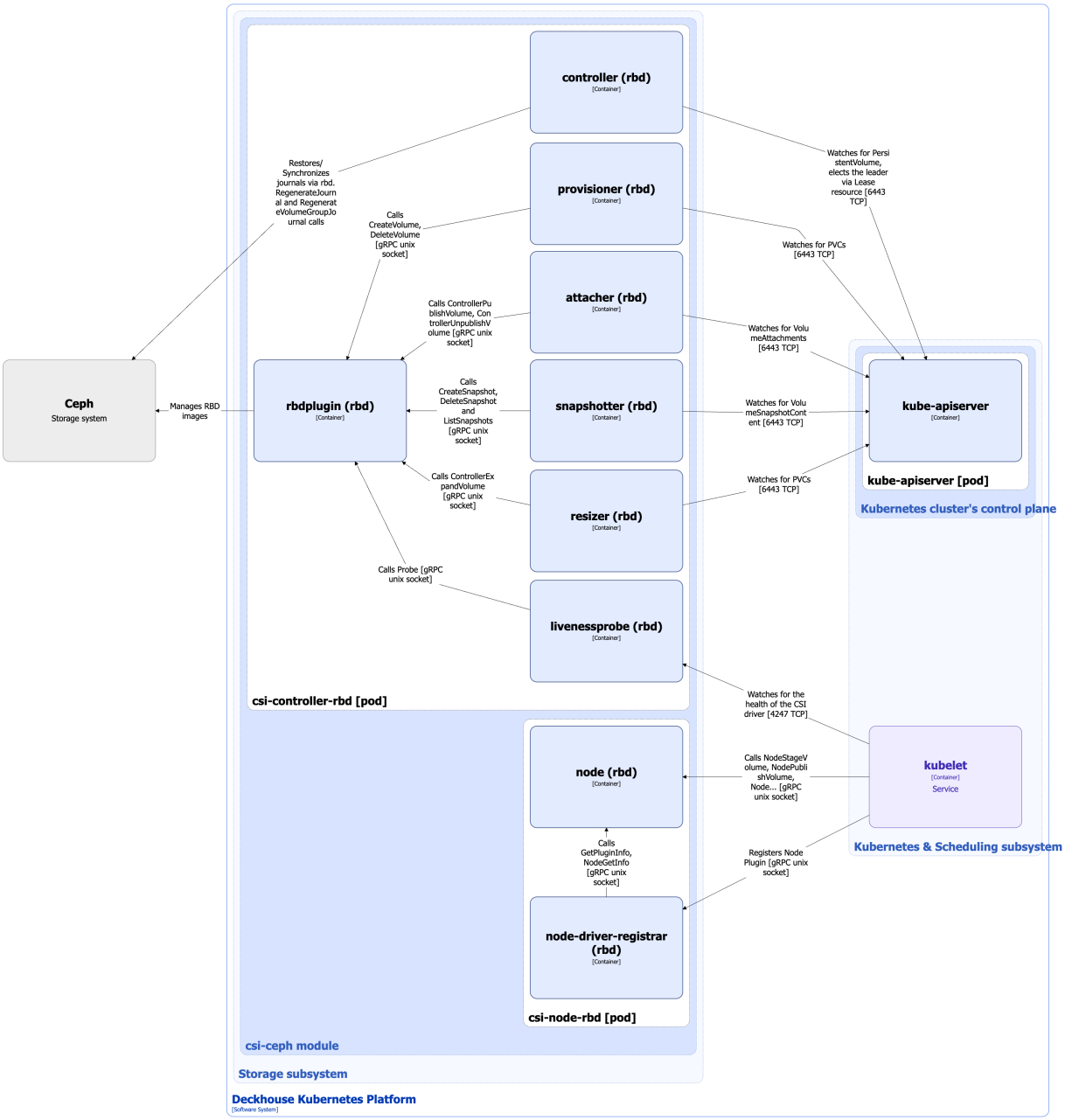
Ceph RBD CSI driver

The `csi-rbd` CSI driver is an implementation of the Container Storage Interface (CSI) standard to manage Ceph RBD (RADOS Block Device) volumes in Deckhouse Kubernetes Platform (DKP).

Driver architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-rbd` CSI driver and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Driver components

The `csi-rbd` CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, attaching and detaching volumes from nodes, and managing snapshots.

   It consists of the following containers:

   - **controller**: Main container that performs background tasks for managing Kubernetes resources related to Ceph.
   - **rbdplugin**: Sidecar container exposes the CSI endpoint over a Unix socket, implements Identity Service and Controller Service according to the CSI specification. It interacts with the Ceph cluster for RBD volumes.
   - **controller sidecar containers**: Kubernetes community-maintained external controllers.

These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DKP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI RPC methods exposed by the rbdplugin container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

External controllers communicate with the rbdplugin container over gRPC via Unix sockets.

Csi-controller includes the following external controllers:

- **Provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`. It also uses `ValidateVolumeCapabilities` to verify compatibility.
- **Attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node and attaches or detaches volumes using the RPC methods `ControllerPublishVolume` and `ControllerUnpublishVolume`.
- **Resizer** (external-resizer): Watches updates to PersistentVolumeClaim resources and expands volumes using the `ControllerExpandVolume` RPC method when a user requests additional storage for a PVC, and the driver supports the `EXPAND_VOLUME` capability.
- **Snapshotter** (external-snapshotter): Works together with the `snapshot-controller` module, watches VolumeSnapshotContent resources, and manages volume snapshots using the RPC methods `CreateSnapshot`, `DeleteSnapshot`, and `ListSnapshots` (if supported by the driver).
- **Livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.

2. **Csi-node** (DaemonSet): Node Plugin running on all cluster nodes and responsible for local volume mount and unmount operations.

   ⚠ The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

   - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
   - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the **node** container occurs over gRPC via a Unix socket.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**: Monitors PersistentVolumeClaim, VolumeAttachment, and VolumeSnapshotContent resources.
2. **Ceph cluster**: Creates and deletes volumes, attaches and detaches volumes from nodes, and manages snapshots.

The following external components interact with the driver:

- Kubelet:
  - Checks the CSI driver livenessProbe.
  - Registers the Node Plugin.
  - Calls the RPC methods `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, `NodeUnpublishVolume`, and `NodeExpandVolume` in Node Plugin.

  Kubelet communicates with Node Plugin via gRPC over a Unix socket.

Additional resources

- Module `snapshot-controller` documentation

Huawei CSI driver

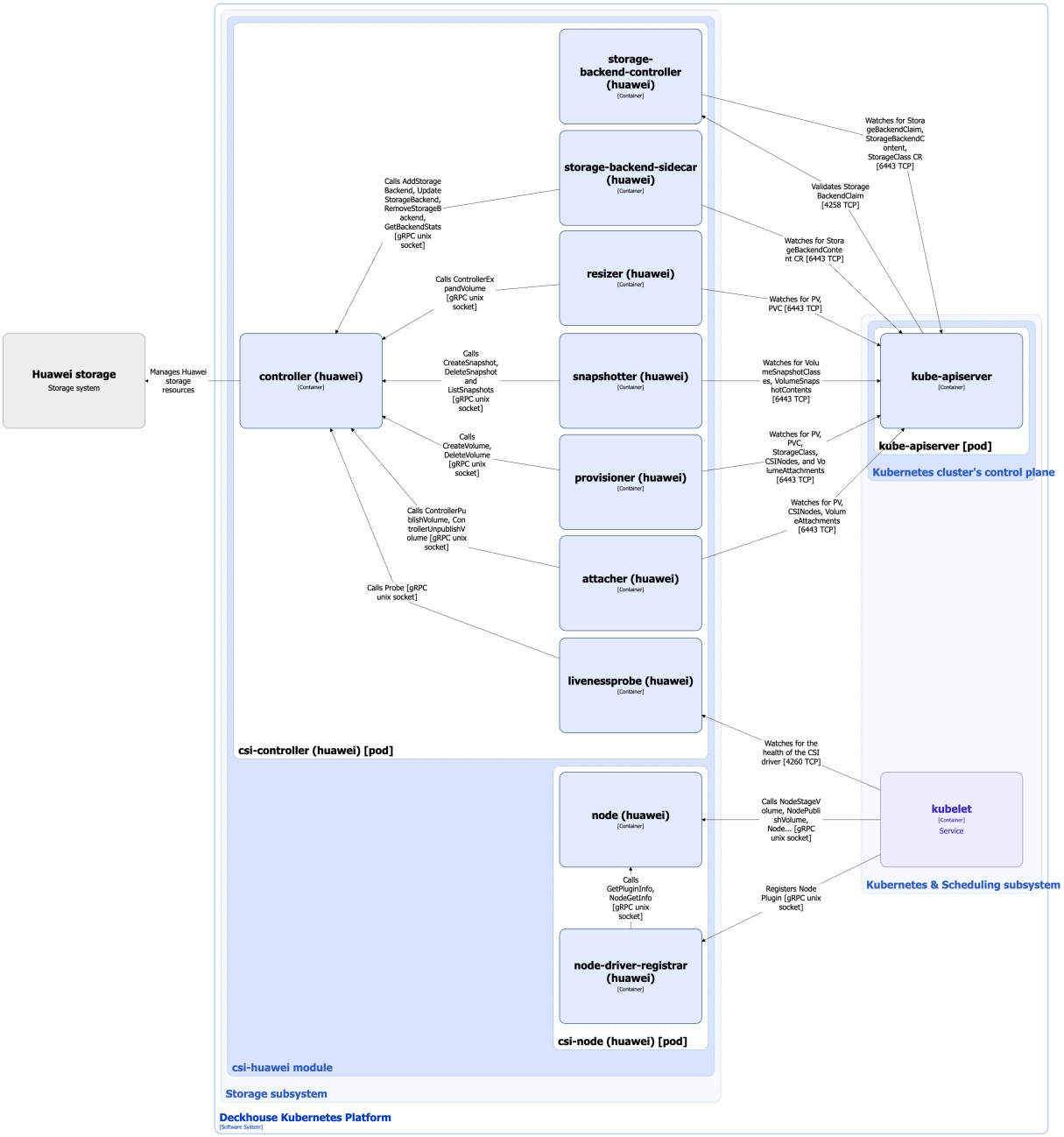
The `csi-huawei` CSI driver is the implementation of the Container Storage Interface (CSI) standard to manage volumes on Huawei storage systems in Deckhouse Kubernetes Platform (DKP).

Driver architecture

i The following simplifications are made in the diagram:

- The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
- Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-huawei` CSI driver and its interactions with other components of DKP are shown in the following diagram:



Driver components

The `csi-huawei` CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, attaching and detaching volumes from nodes, and managing snapshots.

   It consists of the following containers:

   - **controller**: Main container implementing CSI driver functionality (capabilities) through the gRPC services Identity Service and Controller Service according to the CSI specification.
   - **controller sidecar containers**: Kubernetes community-maintained external controllers.

     These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DKP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI driver functions in the controller container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

     External controllers communicate with the controller container over gRPC via Unix sockets.

     The csi-controller includes the following external controllers:

     - **Provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`. It also uses `ValidateVolumeCapabilities` to verify compatibility.

        - **Attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node and attaches or detaches volumes using the RPC methods `ControllerPublishVolume` and `ControllerUnpublishVolume`.
        - **Resizer** (external-resizer): Watches updates to PersistentVolumeClaim resources and expands volumes using the `ControllerExpandVolume` RPC method when a user requests additional storage for a PVC, and the driver supports the `EXPAND_VOLUME` capability.
        - **Snapshotter** (external-snapshotter): Works together with the `snapshot-controller` module, watches VolumeSnapshotContent resources, and manages volume snapshots using the RPC methods `CreateSnapshot`, `DeleteSnapshot`, and `ListSnapshots`.
        - **Livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.
    - **storage-backend-controller**: An additional controller that reconciles the following custom resources:
        - StorageBackendClaim: A request to connect to Huawei storage.
        - StorageBackendContent: Description of the actual connection to Huawei storage.
    - **storage-backend-sidecar**: Sidecar container that provides integration with Huawei storage via `AddStorageBackend`, `UpdateStorageBackend`, `RemoveStorageBackend`, and `GetBackendStats` RPC calls.
2. **Csi-node** (DaemonSet): Node Plugin running on all cluster nodes and responsible for local volume mount and unmount operations.

   ⚠ The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

    - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
    - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the node container occurs over gRPC via a Unix socket.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**:
    - Watches PersistentVolumeClaim and VolumeAttachment resources.
    - Reconciles StorageBackendClaim and StorageBackendContent custom resources.
2. **Huawei storage system**: Creates and deletes volumes, and attaches/detaches volumes to/from nodes.

The following external components interact with the driver:

1. **Kube-apiserver**: Validates StorageBackendClaim custom resources.
2. Kubelet:
    - Checks CSI driver livenessProbe.
    - Registers the Node Plugin.
    - Calls `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, and `NodeUnpublishVolume` RPCs in the Node Plugin.

   Kubelet interacts with the Node Plugin over gRPC via a Unix socket.

Additional resources

- Module `snapshot-controller` documentation

NFS CSI driver

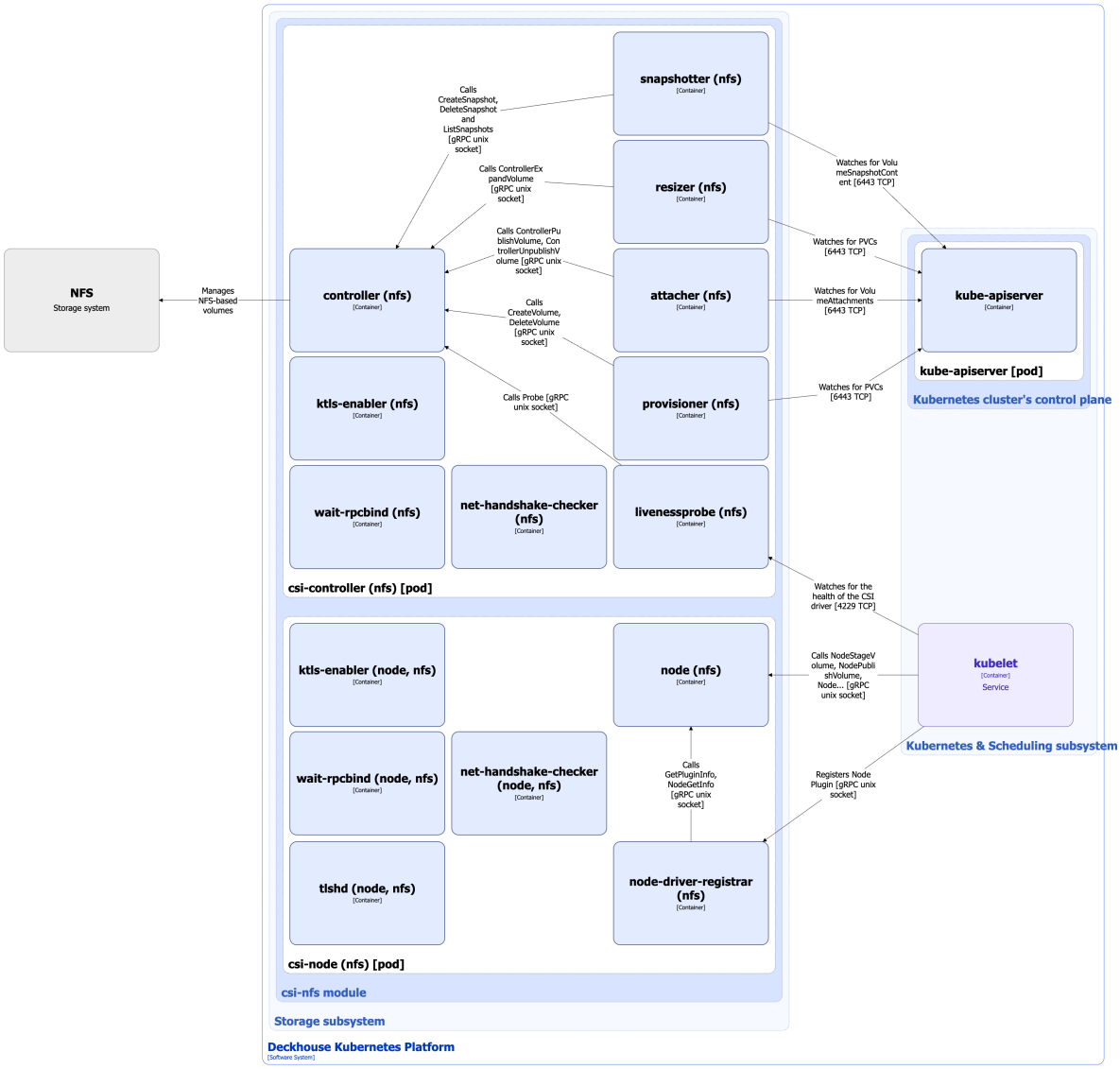
The CSI driver `csi-nfs` is the implementation of the Container Storage Interface (CSI) standard to manage NFS-based volumes in Deckhouse Kubernetes Platform (DKP).

Driver architecture

i The following simplifications are made in the diagram:

- The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
- Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-nfs` CSI driver and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Driver components

The `csi-nfs` CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, attaching and detaching volumes from nodes, and managing snapshots.

   It consists of the following containers:

   - **wait-rpcbind**: Init-container. It waits for when the Unix socket of the `rpcbind` service is ready and accepts connections.
   - **ktls-enabler**: Init-container. It loads and checks Linux kernel modules on the node for RPC-with-TLS support.
   - **net-handshake-checker**: Init-container. It runs `tlshd` one time to check kernel TLS handshake support.
   - **controller**: Main container implementing CSI driver functionality (capabilities) through the gRPC services Identity Service and Controller Service according to the CSI specification.
   - **controller sidecar containers**: Kubernetes community-maintained external controllers.

     These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DKP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI driver functions in the controller container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

     External controllers communicate with the controller container over gRPC via Unix sockets.

     Csi-controller includes the following external controllers:

     - **Provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`. It also uses `ValidateVolumeCapabilities` to verify compatibility.
     - **Attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node

and attaches or detaches volumes using the RPC methods `ControllerPublishVolume` and `ControllerUnpublishVolume`.

- **Resizer** (external-resizer): Watches updates to PersistentVolumeClaim resources and expands volumes using the `ControllerExpandVolume` RPC method when a user requests additional storage for a PVC, and the driver supports the `EXPAND_VOLUME` capability.
- **Snapshotter** (external-snapshotter): Works together with the `snapshot-controller` module, watches VolumeSnapshotContent resources, and manages volume snapshots using the RPC methods `CreateSnapshot`, `DeleteSnapshot`, and `ListSnapshots` (if supported by the driver).
- **Livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.

2. **Csi-node** (DaemonSet): Node Plugin running on cluster nodes labeled with `storage.deckhouse.io/csi-nfs-node` and responsible for local volume mount and unmount operations.

   ⚠ The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

   - **wait-rpcbind**: Init-container. It waits for when the Unix socket of the `rpcbind` service is ready and accepts connections.
   - **ktls-enabler**: Init-container. It loads and checks Linux kernel modules on the node for RPC-with-TLS support.
   - **net-handshake-checker**: Init-container. It runs `tlshd` one time to check kernel TLS handshake support.
   - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
   - **tlshd**: Sidecar container that provides an RPC-with-TLS connection.
   - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the node container occurs over gRPC via a Unix socket.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**: Monitors PersistentVolumeClaim, VolumeAttachment, and VolumeSnapshotContent resources.
2. **NFS storage**:
   - Attaches and detaches NFS-based volumes from nodes.
   - Manages snapshots.

The following external components interact with the driver:

- Kubelet:
  - Checks the CSI driver livenessProbe.
  - Registers the Node Plugin.
  - Calls the RPC methods `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, `NodeUnpublishVolume`, and `NodeExpandVolume` in Node Plugin.

  Kubelet communicates with Node Plugin via gRPC over a Unix socket.

Additional resources

- Module `snapshot-controller` documentation

S3 CSI driver

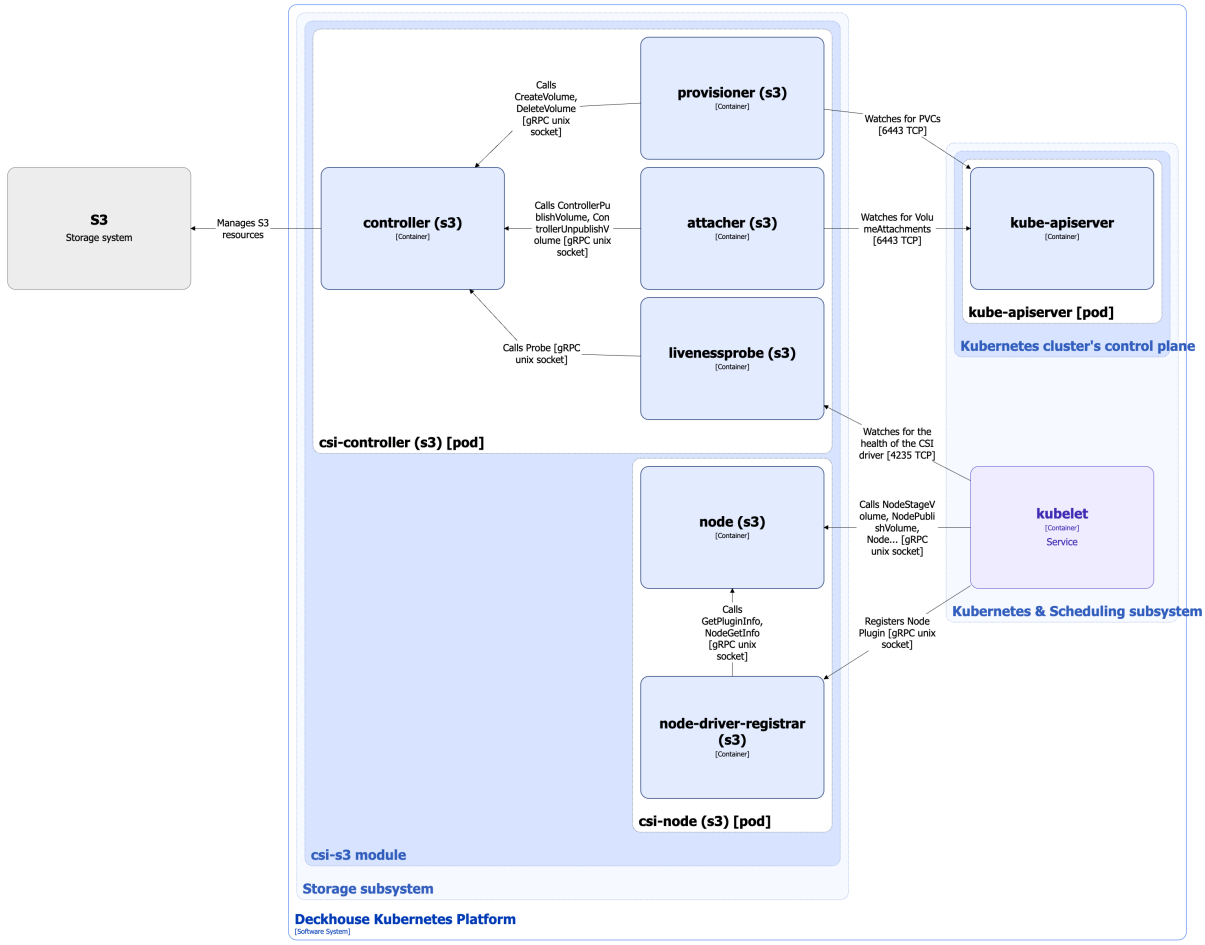
The `csi-s3` CSI driver is the implementation of the Container Storage Interface (CSI) standard to manage S3-based volumes in Deckhouse Kubernetes Platform (DKP).

Driver architecture

ℹ The following simplifications are made in the diagram:

- The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
- Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-s3` CSI driver and its interactions with other components of DKP are shown in the following diagram:



Driver components

The `csi-s3` CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, and attaching and detaching volumes from nodes.

   It consists of the following containers:

   - **controller**: Main container implementing CSI driver functionality (capabilities) through the gRPC services Identity Service and Controller Service according to the CSI specification.
   - **controller sidecar containers**: Kubernetes community-maintained external controllers.

     These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DKP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI driver functions in the controller container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

     External controllers interact with the controller container over gRPC via Unix sockets.

     The csi-controller includes the following external controllers:

     - **Provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`. It also uses `ValidateVolumeCapabilities` to verify compatibility.
     - **Attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node and attaches or detaches volumes using the RPC methods `ControllerPublishVolume` and `ControllerUnpublishVolume`.
     - **Livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.

2. **Csi-node** (DaemonSet): Node Plugin running on all cluster nodes and responsible for local volume mount and unmount operations.

   > ⚠ The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

- **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
- **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the node container occurs over gRPC via a Unix socket.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**: Watches PersistentVolumeClaim and VolumeAttachment resources.
2. **S3 storage**: Creates and deletes volumes, and attaches/detaches volumes to/from nodes.

The following external components interact with the driver:

1. Kubelet:
   - Checks CSI driver livenessProbe.
   - Registers the Node Plugin.
   - Calls `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, and `NodeUnpublishVolume` RPCs in the Node Plugin.

   Kubelet interacts with the Node Plugin over gRPC via a Unix socket.

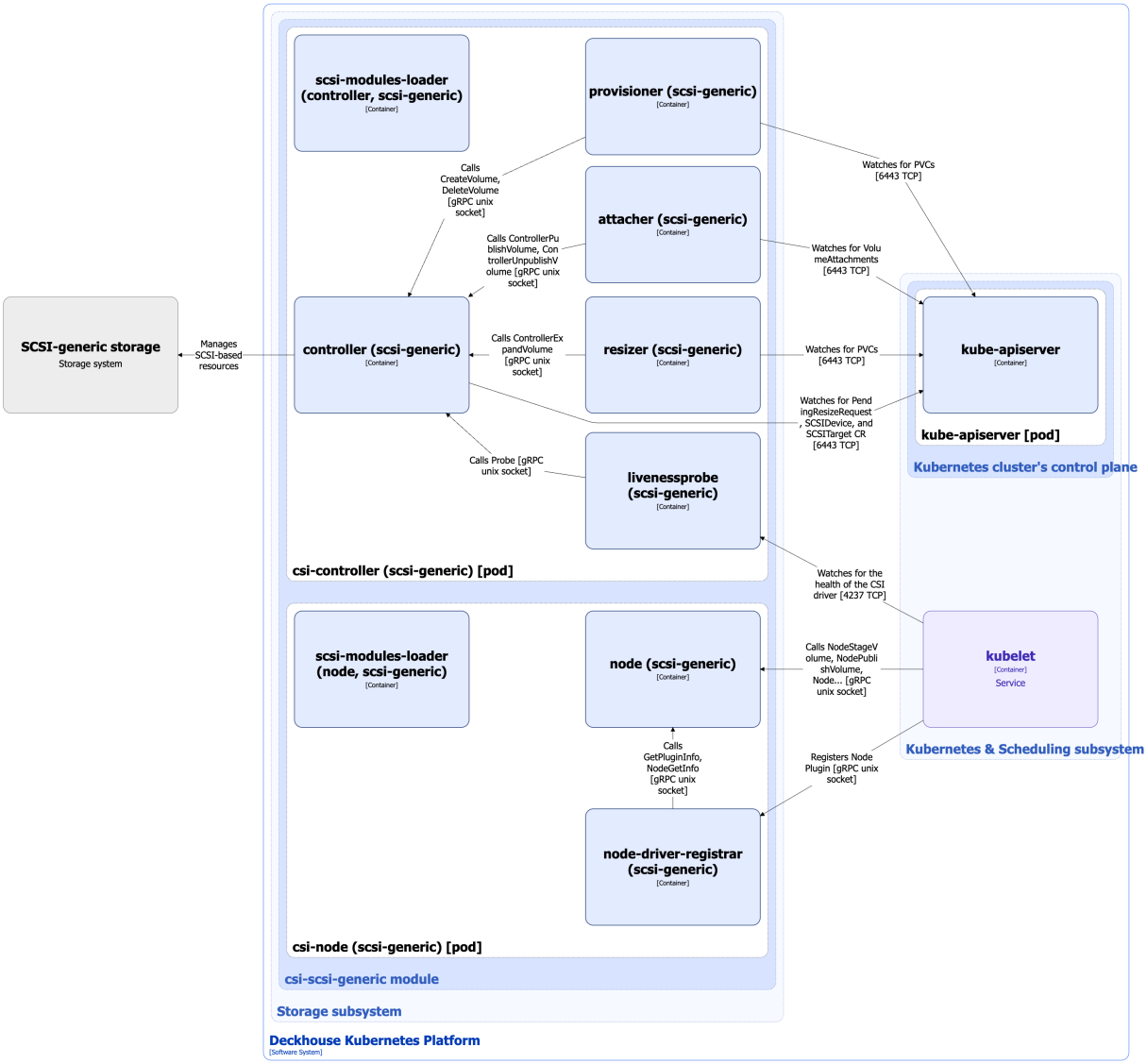
SCSI Generic CSI driver

The `csi-scsi-generic` CSI driver is the implementation of the Container Storage Interface (CSI) standard to manage volumes on SCSI-connected storage systems in Deckhouse Kubernetes Platform (DKP). The CSI driver supports both iSCSI and Fibre Channel (FC) connection technologies.

Driver architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-scsi-generic` CSI driver and its interactions with other components of DKP are shown in the following diagram:



Driver components

The `csi-scsi-generic` CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, and attaching and detaching volumes from nodes.

   It consists of the following containers:

   - **scsi-modules-loader**: Init container that loads the Linux kernel module required for iSCSI (`iscsi_tcp`).
   - **controller**: Main container implementing CSI driver functionality (capabilities) through the gRPC services Identity Service and Controller Service according to the CSI specification.
   - **controller sidecar containers**: Kubernetes community-maintained external controllers.

     These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DKP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI driver functions in the controller container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

     External controllers interact with the controller container over gRPC via Unix sockets.

     The csi-controller includes the following external controllers:

     - **Provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`.
     - **Attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node.
     - **Resizer** (external-resizer): Watches updates to PersistentVolumeClaim resources and expands volumes using the `ControllerExpandVolume` RPC method when a user requests additional storage for a PVC, and the driver supports the `EXPAND_VOLUME` capability. The controller doesn’t expand the Logical Unit Number (LUN) size in the storage system, but updates information in the cluster metadata.
     - **Livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet

restarts the csi-controller pod.

2. **Csi-node** (DaemonSet): Node Plugin running on all cluster nodes and responsible for local volume mount and unmount operations.

   ⚠ The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

   - **scsi-modules-loader**: Init container that loads the kernel module required for iSCSI (`iscsi_tcp`).
   - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
   - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the node container occurs over gRPC via a Unix socket.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**: Watches PersistentVolume, PersistentVolumeClaim, VolumeAttachment, PendingResizeRequest, SCSIDevice, and SCSITarget resources.
2. **SCSI-connected storage systems**: Coordinates the usage of already available SCSI devices, including their binding/cleanup and node attachment.

The following external components interact with the driver:

1. Kubelet:
   - Checks CSI driver livenessProbe.
   - Registers the Node Plugin.
   - Calls `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, and `NodeUnpublishVolume` RPCs in the Node Plugin.

   Kubelet interacts with the Node Plugin over gRPC via a Unix socket.

VMware vSphere CSI driver

A CSI driver (plugin) is used to manage persistent storage volumes in Deckhouse Kubernetes Platform (DKP).

Container Storage Interface (CSI) is a standard interface that unifies access to storage systems and simplifies integration of different storage systems into clusters.

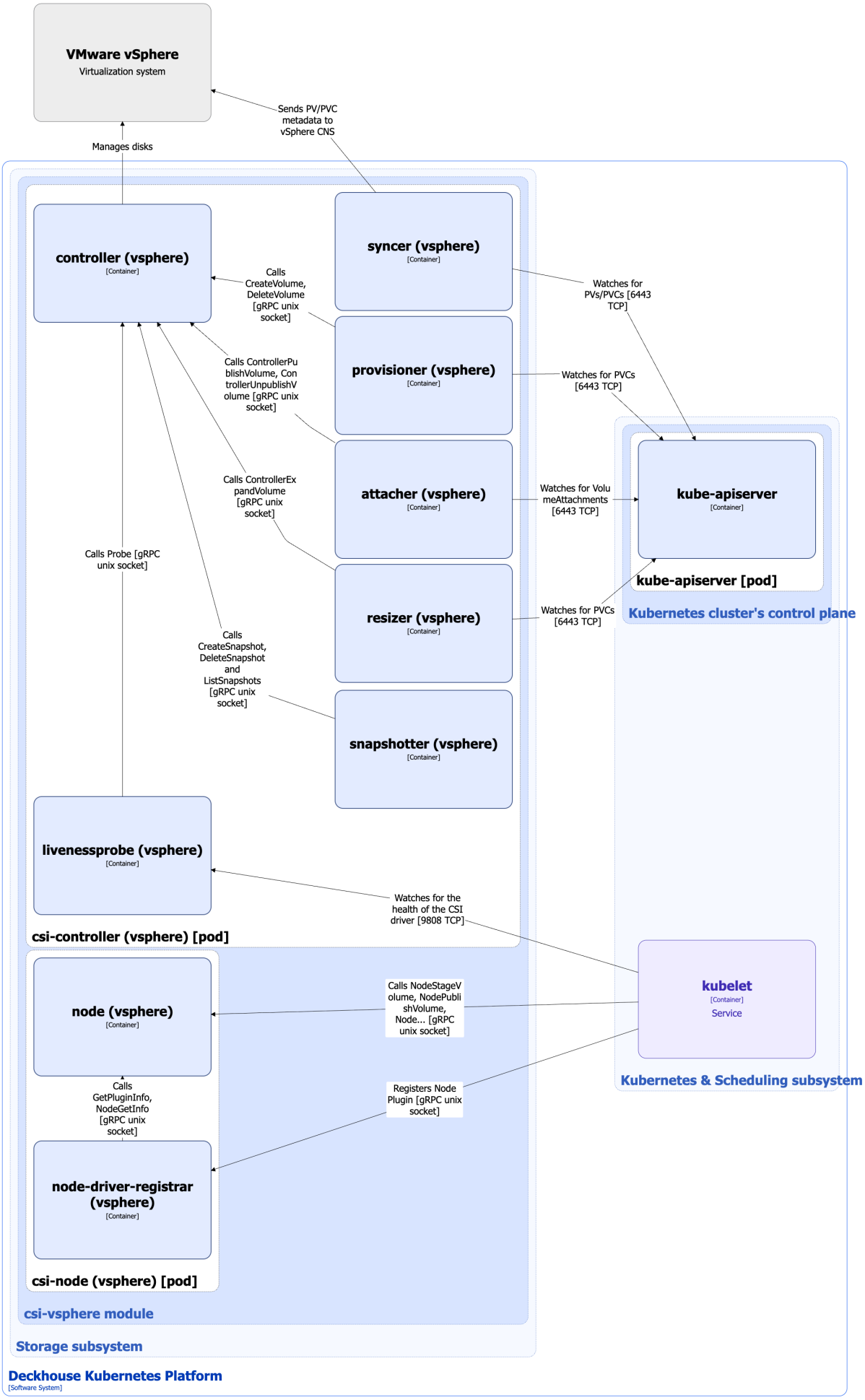
The `cloud-provider-vsphere` module uses `csi-vsphere` (based on vSphere CSI Driver). It differs from the drivers used in other DKP modules by the presence of a syncer component specific to VMware vSphere.

Driver architecture

i The following simplifications are made in the diagram:

- The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
- Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `csi-vsphere` CSI driver and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:



Driver components

The CSI driver consists of the following components:

1. **Csi-controller** (Deployment): Controller Plugin responsible for global volume operations such as creating and deleting volumes, attaching and detaching volumes from nodes, and managing snapshots.

   It consists of the following containers:

   - **controller**: Main container implementing CSI driver functionality (capabilities) through the gRPC services Identity Service and Controller Service according to the CSI specification.
   - **controller sidecar containers**: Kubernetes community-maintained external controllers.

     These controllers are required because the persistent volume controller running in kube-controller-manager (a component of the DKP control plane) does not provide an interface for direct interaction with CSI drivers. External controllers monitor PersistentVolumeClaim resources and call the corresponding CSI driver functions in the controller container. They also perform auxiliary tasks such as retrieving plugin information and capabilities or checking driver health (liveness probe).

     External controllers communicate with the controller container over gRPC via Unix sockets.

     Csi-controller includes the following external controllers:

     - **Provisioner** (external-provisioner): Watches PersistentVolumeClaim resources and calls the RPC methods `CreateVolume` or `DeleteVolume`. It also uses `ValidateVolumeCapabilities` to verify compatibility.
     - **Attacher** (external-attacher): Monitors VolumeAttachment resources after a pod is scheduled to a node and attaches or detaches volumes using the RPC methods `ControllerPublishVolume` and `ControllerUnpublishVolume`.
     - **Resizer** (external-resizer): Watches updates to PersistentVolumeClaim resources and expands volumes using the `ControllerExpandVolume` RPC method when a user requests additional storage for a PVC, and the driver supports the `EXPAND_VOLUME` capability.
     - **Snapshotter** (external-snapshotter): Works together with the `snapshot-controller` module, watches VolumeSnapshotContent resources, and manages volume snapshots using the RPC methods `CreateSnapshot`, `DeleteSnapshot`, and `ListSnapshots` (if supported by the driver).
     - **Livenessprobe**: Monitors the health of the CSI driver through the `Probe` RPC from the Identity Service and exposes the HTTP endpoint `/healthz`, which is checked by kubelet. If *livenessProbe* fails, kubelet restarts the csi-controller pod.
     - **Syncer**: VMware vSphere specific controller, it is responsible for pushing the PersistentVolumes, PersistentVolumeClaims, and pod metadata to Cloud Native Storage (CNS). CNS is a control plane managing various volume types in the VMware vSphere storage system.

2. **Csi-node** (DaemonSet): Node Plugin running on all cluster nodes and responsible for local volume mount and unmount operations.

   > **Warning.** The plugin has privileged access to the filesystem of each node. On Linux, this requires the `CAP_SYS_ADMIN` capability. This is necessary to perform mount operations and interact with block devices.

   It consists of the following containers:

   - **node**: Main container implementing CSI driver functionality through the gRPC services Identity Service and Node Service according to the CSI specification.
   - **node-driver-registrar**: Sidecar container that registers the Node Plugin with kubelet. It calls the RPC methods `GetPluginInfo` and `NodeGetInfo` in the node container to retrieve plugin and node information. Communication with the node container occurs over gRPC via a Unix socket.

Driver interactions

The driver interacts with the following components:

1. **Kube-apiserver**: Monitors PersistentVolumeClaim, VolumeAttachment, and VolumeSnapshotContent resources.
2. **VMware vSphere**:
   - Creates and deletes volumes.
   - Attaches and detaches volumes from nodes.
   - Manages snapshots.
   - Synchronizes PersistentVolumes, PersistentVolumeClaims, and pod metadata with the Cloud Native Storage (CNS).

The following external components interact with the driver:

1. Kubelet:
   - Checks the CSI driver livenessProbe.
   - Registers the Node Plugin.
   - Calls the RPC methods `NodeStageVolume`, `NodeUnstageVolume`, `NodePublishVolume`, `NodeUnpublishVolume`, and `NodeExpandVolume` in Node Plugin.

   Kubelet communicates with Node Plugin via gRPC over a Unix socket.

Additional resources

- Module `cloud-provider-vsphere` documentation
- Module `csi-vsphere` documentation
- Module `snapshot-controller` documentation

Observability subsystem

This section describes the architecture of the Observability subsystem of Deckhouse Kubernetes Platform (DKP).

The Observability subsystem includes the following modules:

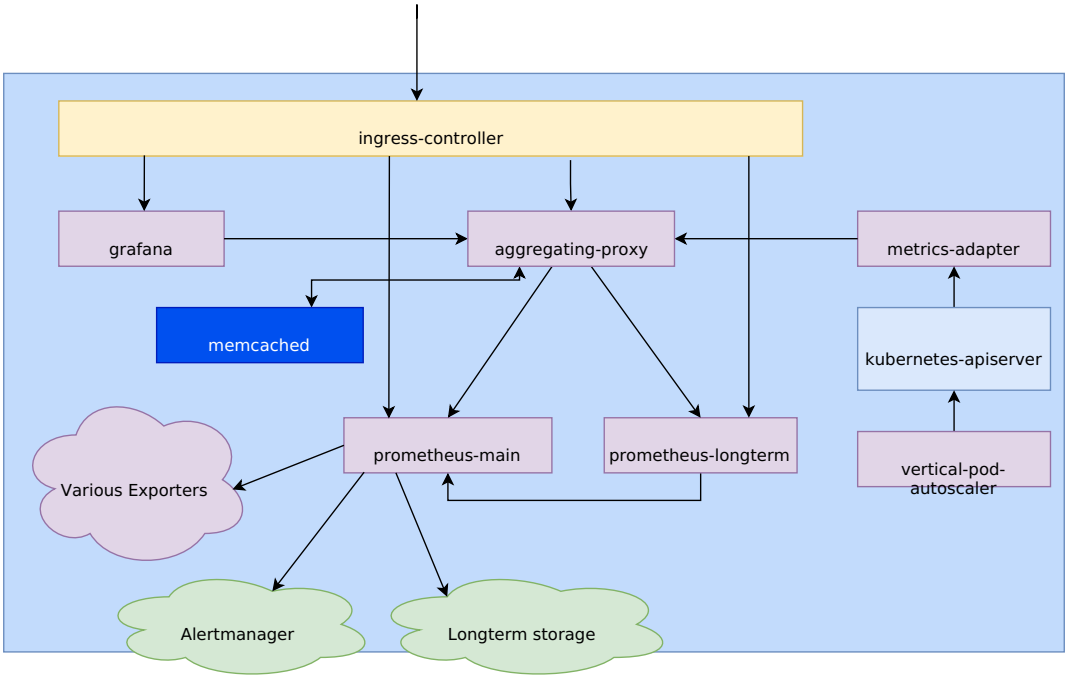
- `prometheus`: Deploys a monitoring stack with predefined settings for DKP and applications, simplifying the initial configuration.
- `operator-prometheus`: Installs Prometheus Operator, which automates the deployment and management of Prometheus instances.
- `prometheus-metrics-adapter`: Allows HPA and VPA autoscalers to use monitoring metrics when making scaling decisions.
- `log-shipper`: Simplifies log collection setup in Kubernetes clusters.
- `loki`: Deploys a short-term log storage system in the cluster based on Grafana Loki.
- `observability`: Extends the functionality of the `prometheus` module and the Deckhouse web UI, providing additional capabilities for flexible management of metrics, dashboards, and alerts, as well as access control mechanisms for them.
- `extended-monitoring`: Enhances cluster monitoring by deploying additional Prometheus exporters that help detect potential issues before they affect service operation.
- `monitoring-custom`: Simplifies monitoring configuration for user applications by requiring only a specific label to be set for the target application.
- `monitoring-deckhouse`: Provides monitoring of DKP components and services.
- `monitoring-kubernetes`: Provides transparent and timely monitoring of all cluster nodes and key infrastructure components.
- `monitoring-kubernetes-control-plane`: Organizes secure metrics collection and provides a basic set of monitoring rules for cluster control plane components.
- `upmeter`: Checks platform availability and cluster component health in real time and displays the results on dedicated dashboards.

The following components are currently described in this section:

- Monitoring architecture in DKP
- `prometheus` module
- `observability` module
- Logging modules

Architecture of monitoring in the Deckhouse Kubernetes Platform

Composition and interaction scheme of monitoring components



Components installed by DKP

Component	Description
prometheus-operator	DKP module responsible for running Prometheus in the cluster.

Component	Description
prometheus-main	Main Prometheus that performs scraping every 30 seconds (can be changed using the `scrapeInterval` parameter). It processes all rules, sends alerts and is the main data source.
prometheus-longterm	Additional Prometheus storing sparse data samples from the main prometheus-main
aggregating-proxy	Aggregating and caching proxy that combines main and longterm into one source. Helps avoid data gaps when one of the Prometheus instances is unavailable.
memcached	In-memory data caching service.
grafana	UI for displaying metrics in dashboard format.
metrics-adapter	Component providing Kubernetes API for accessing metrics. Required for proper VPA operation.
Various exporters	Set of ready-made Prometheus exporters for all necessary metrics: `kube-state-metrics`, `node-exporter`, `oomkill-exporter`, `image-availability-exporter`.
upmeter	Module for assessing DKP component availability.
trickster	Caching proxy that reduces load on Prometheus. Will be deprecated soon.

Prometheus

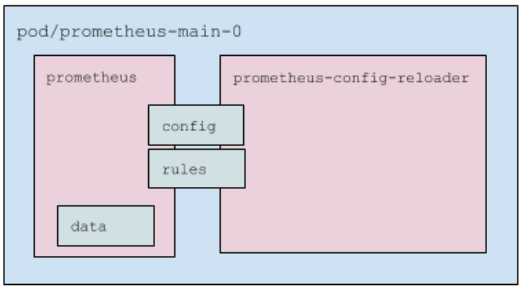
Prometheus collects metrics and executes rules:

- For each *target* (monitoring target) with a given period `scrape_interval`, Prometheus makes an HTTP request to this *target*, receives metrics in its own format in response and saves them to its database.
- Each `evaluation_interval` processes rules (*rules*), based on which:
  - sends alerts;
  - or saves new metrics (result of rule execution) to its database.

Prometheus is installed by the `prometheus-operator` module of DKP, which performs the following functions:

- Defines the following custom resources:
  - `Prometheus`: Defines the installation (cluster) of *Prometheus*.
  - `ServiceMonitor`: Defines how to collect metrics from services.
  - `Alertmanager`: Defines the cluster of *Alertmanagers*.
  - `PrometheusRule`: Defines the list of *Prometheus rules*.
- Monitors these resources, and also:
  - Generates `StatefulSet` with *Prometheus* itself.
  - Creates secrets with configuration files necessary for Prometheus operation (`prometheus.yaml` — Prometheus configuration, and `configmaps.json` — configuration for `prometheus-config-reloader`).
  - Monitors `ServiceMonitor` and `PrometheusRule` resources and updates *Prometheus* configuration files by making changes to secrets based on them.

What's in the Prometheus pod?



- Two containers:
  - `prometheus`: Prometheus itself;
  - `prometheus-config-reloader`: Wrapper that:
    - Monitors changes to `prometheus.yaml` and, if necessary, calls configuration reload to Prometheus (via special HTTP request, see more details below).
    - Monitors PrometheusRules (see more details below) and downloads them and restarts Prometheus as needed.
- Pod uses three volumes:
  - config: Mounted secret (two files: `prometheus.yaml` and `configmaps.json`). Connected to both containers.
  - rules: `emptyDir` that is filled by `prometheus-config-reloader` and read by `prometheus`. Connected to both containers, but in `prometheus` in read-only mode.
  - data: Prometheus data. Mounted only in `prometheus`.

How is Prometheus configured?

- The Prometheus server has *config* and *rule files* (files with rules);

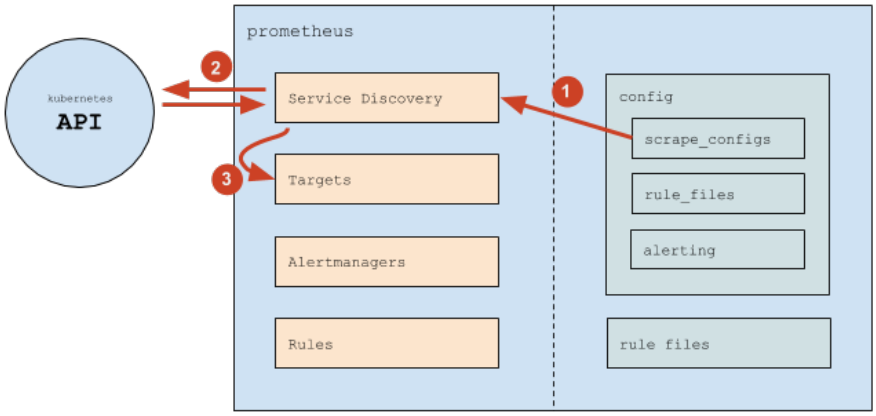
- The `config` has the following sections:
  - `scrape_configs`: Settings for finding *targets* (monitoring targets, see more details in the next section).
  - `rule_files`: List of directories where *rules* that need to be loaded are located:

    ```
    rule_files:
    - /etc/prometheus/rules/rules-0/*
    - /etc/prometheus/rules/rules-1/*
    ```

  - `alerting`: Settings for finding *Alert Managers* to send alerts to. This section is very similar to `scrape_configs`, only the result of its work is a list of *endpoints* to which Prometheus will send alerts.

Where does Prometheus get the list of *targets*?

- In general, Prometheus works as follows:



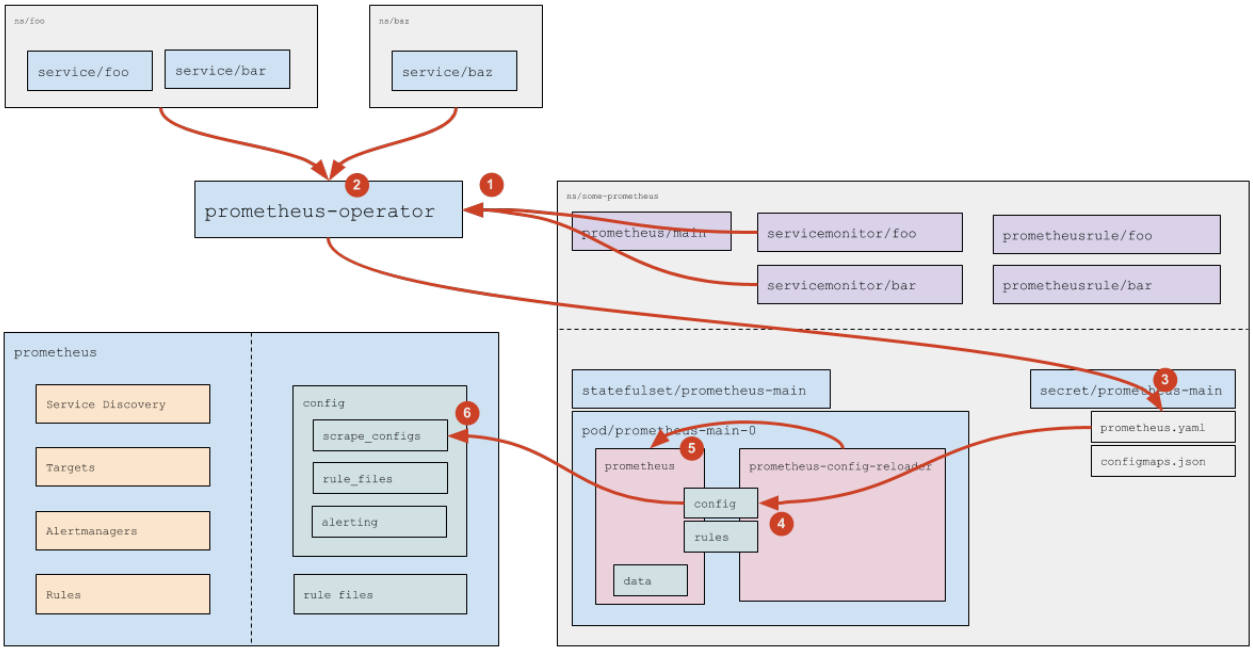
  - **(1)** Prometheus reads the `scrape_configs` section of the config, according to which it configures its internal Service Discovery mechanism.
  - **(2)** The Service Discovery mechanism interacts with the Kubernetes API (mainly — gets endpoints).
  - **(3)** Based on what happens in Kubernetes, the Service Discovery mechanism updates Targets (list of *targets*).

- In `scrape_configs` a list of *scrape jobs* (internal Prometheus concept) is specified, each of which is defined as follows:

```
scrape_configs:
  # General settings
- job_name: d8-monitoring/custom/0   # just the name of the scrape job, shown in the Service
Discovery section
  scrape_interval: 30s               # how often to collect data
  scrape_timeout: 10s                # request timeout
  metrics_path: /metrics              # path to request
  scheme: http                       # http or https
  # Service discovery settings
  kubernetes_sd_configs:              # means that targets are obtained from Kubernetes
  - api_server: null                 # means to use the API server address from environment
variables (which are in every Pod)
    role: endpoints                  # take targets from endpoints
    namespaces:
      names:                        # search for endpoints only in these namespaces
      - foo
      - baz
  # "Filtering" settings (which endpoints to take and which not) and "relabeling" (which
labels to add or remove, on all received metrics)
  relabel_configs:
  # Filter by the value of the prometheus_custom_target label (obtained from the service
associated with the endpoint)
  - source_labels: [__meta_kubernetes_service_label_prometheus_custom_target]
    regex: .+                       # any NON-empty label matches
    action: keep
  # Filter by port name
  - source_labels: [__meta_kubernetes_endpointslice_port_name]
    regex: http-metrics              # matches only if the port is called http-metrics
    action: keep
  # Add job label, use the value of the prometheus_custom_target label of the service, to
which we add the prefix "custom-"
  #
  # The job label is a Prometheus service label:
  #    * it determines the name of the group in which the target will be shown on the targets
page
  #    * and of course it will be on each metric obtained from these targets, so you can
easily filter in rules and dashboards
  - source_labels: [__meta_kubernetes_service_label_prometheus_custom_target]
    regex: (.*)
    target_label: job
    replacement: custom-$1
    action: replace
  # Add namespace label
  - source_labels: [__meta_kubernetes_namespace]
    regex: (.*)
    target_label: namespace
    replacement: $1
    action: replace
  # Add service label
  - source_labels: [__meta_kubernetes_service_name]
    regex: (.*)
    target_label: service
    replacement: $1
    action: replace
  # Add instance label (which will contain the Pod name)
  - source_labels: [__meta_kubernetes_pod_name]
    regex: (.*)
    target_label: instance
    replacement: $1
    action: replace
```

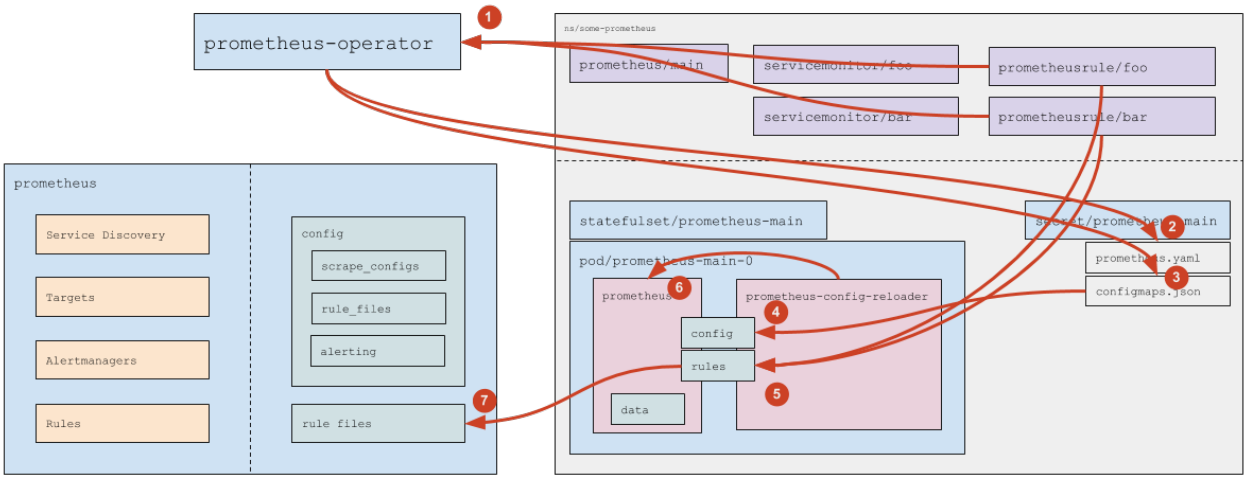
- Thus, Prometheus itself tracks:
  - Addition and removal of Pods (when Pods are added/removed, Kubernetes changes endpoints, and Prometheus sees this and adds/removes *targets*).
  - Addition and removal of services (more precisely endpoints) in specified namespaces.
- Config changes are required in the following cases:
  - Need to add a new scrape config (usually — a new type of services that need to be monitored).
  - Need to change the list of namespaces.

How are Service Monitors processed?



1. Prometheus Operator reads (and also monitors addition/removal/changes) Service Monitors (which specific Service Monitors — specified in the `prometheus` resource itself, see more details in official documentation).
2. For each Service Monitor, if it does NOT specify a specific list of namespaces (specified `any: true`), Prometheus Operator calculates (by accessing the Kubernetes API) a list of namespaces where there are Services (matching the labels specified in the Service Monitor).
3. Based on the read `servicemonitor` resources (see official documentation) and based on the calculated namespaces, Prometheus Operator generates part of the config (the `scrape_configs` section) and saves the config to the corresponding Secret.
4. Using standard Kubernetes means, data from the secret comes to the Pod (the `prometheus.yaml` file is updated).
5. The `prometheus-config-reloader` notices the file change and sends an HTTP request to Prometheus to reload.
6. Prometheus re-reads the config and sees changes in scrape_configs, which it processes according to its own logic (see more details above).

How are custom resources with *rules* processed?



1. Prometheus Operator monitors PrometheusRules (matching the `ruleSelector` specified in the `prometheus` resource).
2. If a new PrometheusRule appears (or an existing one is deleted) — Prometheus Operator updates `prometheus.yaml` (and then the logic exactly corresponding to Service Monitor processing, which is described above, is triggered).
3. Both in case of adding/removing PrometheusRule and when changing the content of PrometheusRule, Prometheus Operator updates the ConfigMap `prometheus-main-rulefiles-0`.
4. Using standard Kubernetes means, data from ConfigMap comes to the Pod
5. The `prometheus-config-reloader` notices the file change and:
   - Downloads changed ConfigMaps to the rules directory (this is `emptyDir`).
   - Sends an HTTP request to Prometheus to reload.
6. Prometheus re-reads the config and sees the changed *rules*.

Architecture of DKP component availability assessment (upmeter)

Availability assessment in DKP is performed by the upmeter module.

Composition of the upmeter module:

- **agent**: Works on master nodes and performs availability probes, sends results to the server.
- **upmeter**: Collects results and maintains an API server for their extraction.

- **front**:
  - **status**: Shows availability level for the last 10 minutes (requires authorization, but it can be disabled);
  - **webui**: Shows a dashboard with statistics on probes and availability groups (requires authorization).
- **smoke-mini**: Maintains continuous *smoke testing* using StatefulSet.

The module sends about 100 metric readings every 5 minutes. This value depends on the number of enabled Deckhouse Kubernetes Platform modules.

Additional resources

- Module `upmeter` documentation

Prometheus module

The `prometheus` module deploys a monitoring stack with predefined settings for Deckhouse Kubernetes Platform (DKP) and provides the collection, storage, and processing of cluster and application metrics.

For more details about the module, refer to the module documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `prometheus` module and its interactions with other components of DKP are shown in the following diagram:

Module components

The module consists of the following components:

1. **Prometheus-main** (StatefulSet): Main Prometheus instance. Prometheus is a monitoring and notification system using a time series database (TSDB). It collects and analyzes application and server performance metrics in real time. Prometheus-main collects metrics from configured monitoring objects every 30 seconds. You can use the `scrapeInterval` parameter to change this value.

   Prometheus-main can use the original (“vanilla”) Prometheus or Deckhouse Prom++, a high-performance open-source fork of Prometheus designed to reduce memory consumption while maintaining full compatibility with the original project. The module uses Deckhouse Prom++ by default. It is possible to switch from Deckhouse Prom++ to the original Prometheus. In this case, migration of write-ahead log (WAL) data will be required, since Deckhouse Prom++ uses its own WAL log format. Migration is performed automatically using the prompptool init container.

   Prometheus-main is the main data source. It collects metrics, processes configured rules, and sends alerts according to set configuration. The Prometheus Operator creates the Prometheus instance and its configuration based on following custom resources:

   - Prometheus: Describes a Prometheus installation (cluster).
   - ServiceMonitor: Defines metrics collection settings from a set of services.
   - PrometheusRule: Contains a set of Prometheus rules.

   Prometheus Operator watches Prometheus resources and, for each one, generates:

   - StatefulSet with Prometheus itself.
   - Secret containing `prometheus.yaml` (the main configuration file) and `configmaps.json` (the configuration file for the config-reloader container, described below). The `prometheus-main` Secret is mounted in prometheus-main Pod and is used by the config-reloader container.

   Prometheus Operator also watches ServiceMonitor and PrometheusRule resources and, based on them, updates the configuration (`prometheus.yaml` and `configmaps.json`) in the described above Secret.

   For more details about the working of Prometheus Operator, refer to the `operator-prometheus` module documentation section.

   For more details about the prometheus-main operation, refer to the section describing the architecture of the monitoring in DKP.

   Prometheus-main consists of the following containers:

   - **init-config-reloader**: Init container that performs a single run of config-reloader to load the Prometheus configuration.
   - **prompptool**: Init container that performs automatic WAL data migration in case of switching from Deckhouse Prom++ to the original Prometheus and vice versa.
   - **config-reloader**: Sidecar container that monitors changes in the `prometheus.yaml` configuration file and, if needed, triggers a Prometheus configuration reload (via a special HTTP request to the `/-/reload` endpoint). Config-reloader is a utility from the Prometheus Operator open-source project.
   - **prometheus**: Main container.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides

secure access to the Prometheus server. It is an open-source project.

2. **Prometheus-longterm** (StatefulSet): An additional Prometheus instance that scrapes the data of the primary Prometheus instance (`prometheus-main`). This allows users to view and analyze historical trends over a long period of time. Prometheus-longterm receives data due to a configured federation mechanism with the primary Prometheus instance.

   The original Prometheus or Deckhouse Prom++ can be used in prometheus-longterm too. Prometheus-longterm has the same set of containers as in prometheus-main, as well as operation principles.

3. **Grafana-v10**: Optional Grafana component that provides a web interface for visualizing monitoring data. Grafana displays dashboards supplied with DKP modules. Grafana could work in High Availability mode, does not store state, and is configured with custom resources. Grafana is enabled by default, but it can be disabled using the `settings.grafana.enabled` parameter.

   > **i** Grafana-v10 will be disabled in the future DKP releases. To view the monitoring dashboards, use the Deckhouse web UI.

   It consists of the following containers:

   - **dashboard-provisioner**: Sidecar container that watches GrafanaDashboardDefinition custom resources and when a new resource appears, it adds to the Grafana folder the dashboards described there.
   - **grafana**: Main container. It is an open-source project.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to the Grafana server and its metrics (described above).

4. **Aggregating-proxy**: Component that performs metrics caching, data collection from several Prometheus instances (if they are in High Availability mode), data deduplication, and query calculation.

   It consists of the following containers:

   - **wait-memcached**: Init container that waits the memcached component to be available over the network. Aggregating-proxy uses memcached to cache metrics in RAM.
   - **mimir**: Sidecar container that works with the memcached component to optimize queries and cache data. If there is no data in the cache, mimir forwards the request to the prometheus-main component via promxy, that is another sidecar container. It is an open-source project.
   - **promxy**: Sidecar container that proxies requests to the prometheus-main component and provides a single endpoint to access data of multiple Prometheus instances. It is an open-source project.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to the mimir containers (queries to the Prometheus server and container metrics) and promxy (queries to container metrics). It’s described above in detail.

5. **Memcached** (StatefulSet): Component used by aggregating-proxy for caching Prometheus metrics. Memcached is a software that implements a service for caching data in RAM to speed up Prometheus metrics query execution.

   It consists of the following containers:

   - **memcached**: Main container. It is an open-source project.
   - **exporter**: Sidecar container that exports memcached container metrics. Exporter collects memcached container metrics via a network connection, as well as from the memcached process’s PID file. It is an open-source project.

6. **Trickster**: Caching proxy server that reduces the load on Prometheus. It is used for caching and proxying prometheus-longterm requests. In the future DKP releases, it’s expected to become deprecated and will no longer be supported.

   It consists of the following containers:

   - **trickster**: Main container. It is an open-source project.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to the proxy server and its metrics (described above).

   > **i** In the future DKP releases, alerts-receiver will be removed from the `prometheus` module. The Alertmanager component of the `observability` module will be used to receive all alerts.

7. **Alerts-receiver**: A server compatible with Alertmanager API. Alerts-receiver receives basic alerts from prometheus-main, creates the ClusterAlert custom resources based on them, updates their statuses and deletes them if the alert is no longer active. The ClusterAlert custom resources are used to inform DKP users on active alerts and are displayed in the Deckhouse web UI. Alerts-receiver is developed by Flant. It consists of one container.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Watches the PrometheusRule and GrafanaDashboardDefinition custom resources.
   - Manages the ClusterAlert custom resources.
   - Authorizes requests for metrics of module components.
2. **Alertmanager**: Sends custom alerts.

The Prometheus instance, which is part of the module, collects metrics from all DKP components:

- Components of modules
- Components of cluster control plane
- Exporters that collect metrics of cluster hardware resources utilization
- Exporters that collect Kubernetes resources metrics
- User applications (additional configuration is required)

To prevent complicating the diagram with a large number of relations, Prometheus interactions related to the collection of metrics from DKP components are omitted.

The following external components interact with the module:

1. **Ingress-controller** (shown on the diagram as “controller nginx”): Forwards users requests to Grafana.

Fault tolerance and High Availability (HA) monitoring mode

The `prometheus` module provides built-in fault tolerance for all its key components. All monitoring services (Prometheus servers, storage systems, proxies, and other important components) are deployed in multiple copies by default. This ensures that in the event of a failure of a separate instance, the service will continue to work without loss of data and availability.

Prometheus, the main component of metric collection, runs in at least two copies (if there are enough nodes in the cluster). All Prometheus instances use the same configuration and receive the same data. To ensure seamless operation in case of failure of one of the copies, a special component, the aggregating-proxy, is used to access Prometheus. It allows you to combine metrics from several Prometheus instances and always return the most complete and up-to-date data, even if one of the copies is temporarily unavailable.

Additional resources

- Module `console` documentation
- Module `observability` documentation
- Module `operator-prometheus` documentation
- Module `prometheus` documentation

Observability module

The `observability` module extends the functionality of the `prometheus` module and the Deckhouse web UI, providing additional capabilities for flexible management of metrics, dashboards, and alerts, as well as access control mechanisms for them.

Module capabilities:

- **Dashboard management**: Allows users to add their own dashboards in Grafana format.
- **Alerting and recording rule group management**: Allows creating and configuring custom alerting and recording rule groups.
- **Notification configuration**: Allows configuring notification channels (Telegram, Slack, email, and webhooks), notification policies, and disabling notifications when needed.
- **Alert management**: Provides information about active alerts and stores the history of resolved alerts.
- **Standard data sources provisioning**.
- **Custom data sources support**: The module allows you to add custom data sources in addition to the provided set of standard data sources.

For more details about the module, refer to the module documentation.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `observability` module and its interactions with other components of Deckhouse Kubernetes Platform (DKP) are shown in the following diagram:

Module components

The module consists of the following components:

1. **Observability-controller**: Consists of a single **observability-controller** container and manages the lifecycle of most of the module’s custom resources, such as ObservabilityMetricsRulesGroup, ObservabilityNotificationChannels, ObservabilityNotificationSilence, and others. For complete list of resources managed by the module, refer to the “Custom resources” section in the module documentation.
2. **Observability-webhook**: Consists of a single **observability-webhook** container that implements a webhook server, which is used to validate and mutate Kubernetes API resources through the Validating/Mutating Admission Controllers mechanisms.
3. **Alert-kube-api**: Consists of a single **alert-kube-api** container and implements Kubernetes Extension API Server that extends Kubernetes API with ObservabilityAlert и ClusterObservabilityAlert custom resources. Alert-kube-api

allows you to request alerts as custom resources using the Alertmanager component as a backend and caches them in memory for quick access.

4. **Alertmanager**: Receives alerts from prometheus-main component of the `prometheus` module, processes and sends them to the end recipients. DKP supports sending alerts through the following delivery channels:

   - `Email`
   - `Telegram`
   - `Slack`
   - `Webhook`

   It consists of the following containers:

   - **alertmanager**: Main container. A fork of the original Alertmanager made by Flant that supports multitenancy, which allows dividing alerts by system (cluster) and project ones. The delivery of alerts (policies, channels, silencers) is also multitenant, allowing individual configuration of channels and delivery policies in different projects.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the Alertmanager API endpoint. It is an open-source project.

5. **Grafana**: A component that provides a web interface for visualizing monitoring data. A fork of Grafana made by Flant is used in the module. The modified Grafana has advanced features, such as separate access to metrics and dashboards according to multitenancy. Grafana dashboards of the `observability` module are integrated into the Deckhouse web UI (monitoring system management from one window).

   It consists of the following containers:

   - **grafana**: Main container.
   - **grafana-kube-storage**: Sidecar container that implements a backend for grafana container and providing management of Dashboard resources and reading Datasource resources of the Grafana component API. These resources allow you to view and manage dashboards within namespaces (projects), as well as to connect custom data sources.
   - **nginx**: Sidecar container that is an NGINX proxy server, which is used to publish static files. It is an open-source project.

6. **Label-enforcer**: A component that authorizes and proxies user requests to metrics (prometheus-main via the label-proxy service) and logs (loki via the logs-gateway service) sources, specified in the Datasource resources of grafana component API. Label-enforcer verifies RBAC access to monitoring data based on user rights, retrieves a list of available namespaces, and enriches requests with labels to filter requested data within user namespaces. For more information about metrics access control, refer to the module documentation. Label-enforcer handles both metrics read and write requests.

   It consists of a single container named **enforcer**.

7. **Opagent** (DaemonSet): An agent designed to collect metrics from both the operating system and the application software installed on the servers. OpAgent is developed by Flant for the Deckhouse Observability Platform (DOP) based on Okagent (Okmeter agent), that is also a part of the Okmeter monitoring system.

   In the `observability` module, opAgent connects to managed services, such as `managed-postgres`, `managed-memcached`, `managed-kafka`, and others (the list of supported managed services is constantly expanding), collects metrics from them, and sends them to prometheus-main component of the `prometheus` module. If the `observability-platform` module is enabled, opAgent collects metrics from the cluster nodes and sends them to DOP.

   OpAgent sends the collected data using the Prometheus Remote Write protocol:

   - Metrics from managed services: To Prometheus via label-enforcer.
   - Metrics from cluster nodes: To DOP.

   It consists of a single container named **opagent**.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Authorizes requests for monitoring data.
   - Manages module custom resources.
2. **Prometheus**: Uses it as a data source and destination.
3. **Loki**: Uses it as a data source.
4. **Alerts receivers**: Sends alerts.
5. DOP: Uses it as a data destination (metrics from cluster nodes).

The following external components interact with the module:

1. **Kube-apiserver**:
   - Validates and mutates module custom resources (with validating and mutating webhooks).
   - Forwards to alert-kube-api requests for ObservabilityAlert and ClusterObservabilityAlert custom resources.
2. **Prometheus-main**: Sends alerts to alertmanager.
3. **Deckhouse web UI**: Uses Grafana for visualizing monitoring data.

Additional resources

- Module `console` documentation
- Module `managed-kafka` documentation
- Module `managed-memcached` documentation
- Module `managed-postgres` documentation
- Module `observability` documentation
- Module `prometheus` documentation

Upmeter module

The `upmeter` module continuously checks Deckhouse Kubernetes Platform (DKP) availability and cluster component health. Probe results are displayed on dashboards.

To learn more about module settings and usage examples, see the `upmeter` configuration page.

Module architecture

> **i** The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica per Pod is shown in the diagram.

The level-2 C4 architecture of the `upmeter` module and its interactions with other DKP components are shown below.

Numbers in the diagram show the user request flow to the status and webui components. Steps 1, 2, and 3 handle requests through Ingress NGINX Controller with mandatory user authentication in the platform-wide auth system provided by the `user-authn` module. For details, see the `user-authn` architecture section.

Module components

The module includes the following components:

1. **Upmeter** (StatefulSet) is a controller that:
   - Watches the custom resource Downtime and calculates DKP component availability excluding downtime intervals defined in this resource.
   - Stores DKP component availability metrics in a local SQLite database.
   - Receives and processes DKP component probe data.
   - Handles API requests for platform availability data.
   - Watches the custom resource UpmeterRemoteWrite and sends probe results to the endpoint defined in that resource by using the Prometheus Remote Write protocol.

   It includes the following containers:
   - **upmeter**: Main container.
   - **migrator**: Init container that applies SQL migrations to the component SQLite database.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC for secure access to the upmeter API.
2. **Upmeter-agent** (DaemonSet) runs on master nodes and regularly executes the following probe groups:
   - Control-plane: API server availability and controller health checks.
   - Deckhouse: DKP cluster health and `deckhouse` module controller checks.
   - Extensions: Checks that every extension has at least one `Ready` Pod.
   - Load-balancing: Checks availability of network load balancing services.
   - Monitoring-and-autoscaling: Checks that the Observability subsystem is healthy and gathers metrics from system components.
   - Nginx: Checks that every Ingress NGINX Controller has at least one `Ready` Pod.
   - Nodegroups: Checks the number of `desired` nodes in each NodeGroup.
   - Synthetic: Checks network connectivity between cluster nodes with HTTP requests to smoke-mini-[a-e].

   The control-plane group includes the following probes:
   - Apiserver: Upmeter-agent checks Kubernetes API availability.
   - Basic-functionality: Upmeter-agent checks basic Kubernetes API behavior through the ConfigMap lifecycle.
   - Namespace: Upmeter-agent creates the `upmeter-probe-namespace` namespace and removes it after validation.
   - Scheduler: Upmeter-agent creates a Pod named `upmeter-probe-scheduler`, verifies that it is scheduled to any cluster node, and removes it.
   - Controller-manager: Upmeter-agent creates a StatefulSet named `upmeter-probe-controller-manager` with an intentionally missing container in the Pod spec, verifies that the target Pod is created and reaches the expected state, and then removes the StatefulSet.
   - Cert-manager: Upmeter-agent creates a self-signed Certificate named `upmeter-probe-cert-manager`, verifies that cert-manager created the related Secret, and then removes the Certificate and Secret.

   The following flow is used to check the deckhouse controller:
   - Upmeter-agent creates or updates the custom resource UpmeterHookProbe.
   - The deckhouse controller watches this resource and runs a hook to update it.
   - Upmeter-agent also watches UpmeterHookProbe and validates changes.

You can disable probes or probe groups by using the `.spec.settings.disabledProbes` parameter.

Upmeter-agent sends collected probe results to upmeter with an HTTP request `POST /downtime`.

It includes the following containers:

- **agent**: Main container.
- **chown-volume-data**: Init container that sets required permissions for the `/var/lib/upmeter/agent` directory on a cluster node.
- **migrator**: Init container that applies SQL migrations to the component SQLite database.

3. **Smoke-mini-[a-e]** (StatefulSet) is used for synthetic connectivity checks. It includes a single **smoke-mini** container. When upmeter-agent sends a request, smoke-mini-[a-e] instances send requests to each other and to the cluster DNS service, then return probe results.

   When the `upmeter` module is installed, the deckhouse controller from the `deckhouse` module registers a hook that distributes StatefulSet instances across different cluster nodes when possible. After that, it rebalances one StatefulSet every minute to another node.

4. **Status** (Deployment) includes a single **status** container and serves a web page with the current availability status of all DKP components.

5. **Webui** (Deployment) includes a single **webui** container and serves a dashboard with per-component availability history.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Manages custom resources UpmeterRemoteWrite and UpmeterHookProbe.
   - Processes the Downtime custom resources.
   - Authorizes requests to upmeter.
   - Creates, validates, and deletes standard resources: Pod, StatefulSet, Namespace, and Secret, and the custom resource Certificate.
2. **External metric storage systems**: Sends probe results with the Prometheus Remote Write protocol.

The following external components interact with the module:

1. **Prometheus-main**: Uses monitoring rules and metrics related to the upmeter module.
2. **Controller nginx**: Forwards external user requests to the module web interface.

Additional resources

- Module `upmeter` documentation

Logging modules

Log-shipper module

The `log-shipper` module simplifies log collection configuration in Kubernetes clusters. It allows collecting logs both from applications running in the cluster and from the cluster nodes, and then forwarding them to any log storage system — either internal or external (for example, Loki, Elasticsearch, and others).

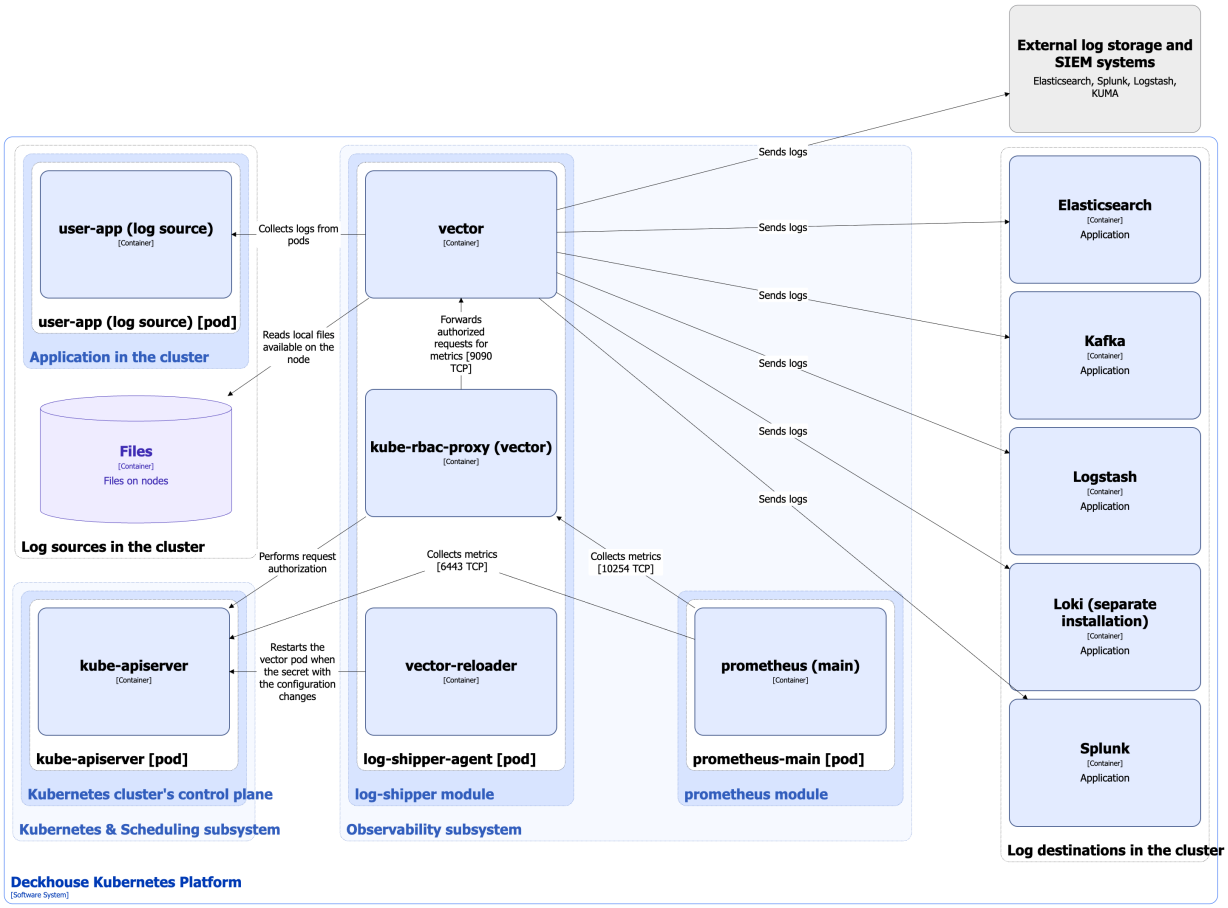
> **i** Deckhouse Kubernetes Platform (DKP) provides integration with log storage systems. The storage systems themselves must be deployed and configured by the user.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `log-shipper` module and its interactions with other components of DKP are shown in the following diagram:



Module components

The module consists of a single component:

- **log-shipper-agent** (DaemonSet): Separate instance of log-shipper-agent runs on each cluster node and includes the following containers:
  - **vector**: Logging agent based on Datadog Vector.

    It is configured using the custom resources ClusterLogDestination, ClusterLoggingConfig, and PodLoggingConfig.
  - **vector-reloader**: Sidecar container running Reloader. It monitors changes to the configuration Secret. When changes are detected, it validates the updated configuration and restarts vector to apply the updates.
  - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to agent metrics.

Module interactions

The module interacts with the following components:

1. **Log sources in the cluster**:
   - Applications running in the cluster — the module collects logs from pods.
   - Files — the module reads local files on cluster nodes.
2. **Log destinations**:
   - Internal log storage systems.
   - External log storage systems and SIEM platforms.

   Both internal and external destinations may include Elasticsearch, Kafka, Logstash, Loki, and Splunk.
3. **Kube-apiserver**:
   - Authorizes requests for metrics.
   - Monitors changes to the Secret containing the vector configuration.

The following external components interact with the module:

1. **Prometheus-main** — collects metrics from log-shipper-agent.

Loki module

In Kubernetes, system logs stored on nodes are retained only for a limited time and may be lost during node restarts or updates. The `loki` module deploys an internal short-term log storage system in the cluster based on Grafana Loki.

Module capabilities:

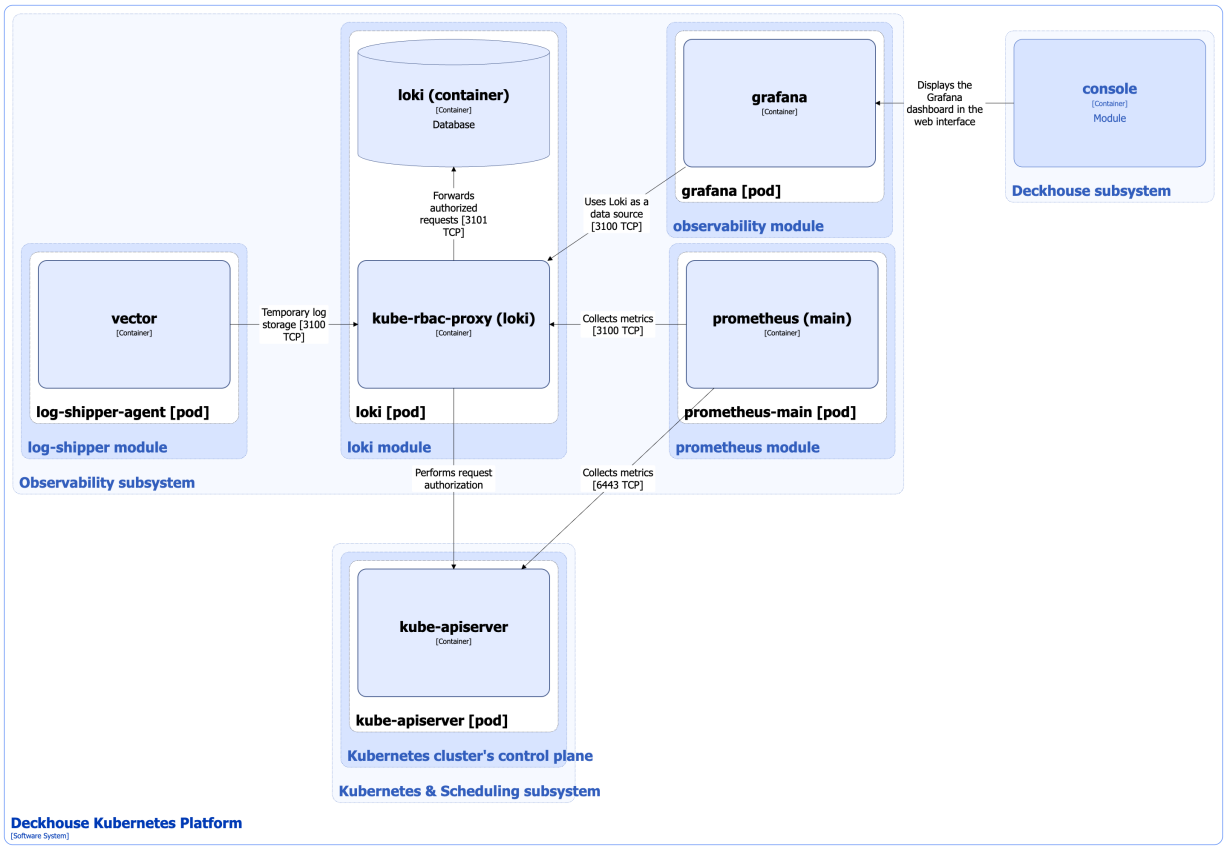
- System logs are automatically sent to Loki without additional configuration.
- Logs can be accessed through Grafana and the Deckhouse web UI (the `console` module).

> ⚠ The short-term log storage based on Grafana Loki does not support high availability (HA) mode. For long-term storage of important logs, use external storage systems supported by the `log-shipper` module.

For more details about module configuration and usage examples, refer to the corresponding documentation section.

Module architecture

The Level 2 C4 architecture of the `loki` module and its interactions with other components of DKP are shown in the following diagram:



Module components

The module consists of a single component:

- **loki**: *StatefulSet* with a single replica **loki-0**, which includes the following containers:
  - **loki**: Container running Grafana Loki.
  - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to loki and its metrics.

Module interactions

The following external components interact with the module:

- **vector**: Sends logs from system components to loki.
- **console** (the Grafana instance embedded in the web UI): Uses loki as a data source for log visualization and analysis.
- **prometheus-main**: Collects loki metrics.

Additional resources

- Module `log-shipper` documentation
- Module `loki` documentation

Managed Services subsystem

This subsection describes the architecture of the Managed Services subsystem of Deckhouse Kubernetes Platform (DKP). Managed Services subsystem automates the deployment, scaling, backup, and updating of managed services in DKP.

The Managed Services subsystem includes the following modules

- `managed-cassandra`: Manages Cassandra clusters.
- `managed-clickhouse`: Manages ClickHouse instances.
- `managed-hive-metastore`: Manages Hive Metastore clusters.
- `managed-kafka`: Manages Kafka instances.
- `managed-memcached`: Manages Memcached instances.
- `managed-opensearch`: Manages OpenSearch instances.
- `managed-postgres`: Manages PostgreSQL clusters.
- `managed-starrocks`: Manages StarRocks clusters.
- `managed-trino`: Manages Trino clusters.
- `managed-valkey`: Manages Valkey clusters.

Additional resources

- Module `managed-cassandra` documentation
- Module `managed-clickhouse` documentation
- Module `managed-hive-metastore` documentation
- Module `managed-kafka` documentation
- Module `managed-memcached` documentation
- Module `managed-postgres` documentation

Managed-memcached module

The `managed-memcached` module simplifies the deployment and management of Memcached instances in Deckhouse Kubernetes Platform (DKP). It provides:

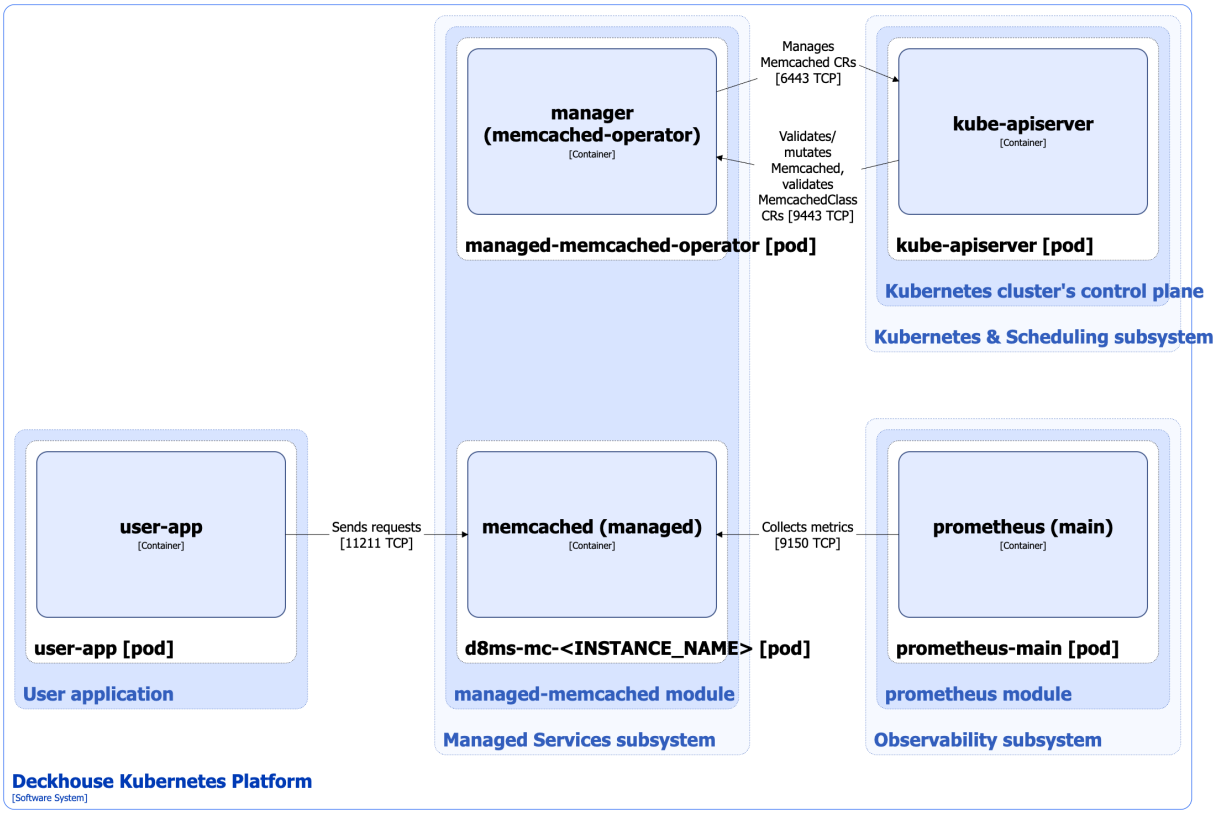
- **Automated Deployment**: Deploy Memcached instances with a simple YAML configuration.
- **High Availability**: Support for both standalone (Standalone) and group (Group) deployments.
- **Configuration Management**: Flexible configuration with validation and constraints through MemcachedClass.
- **Resource Management**: Automatic resource allocation and scaling.
- **Monitoring**: Built-in status tracking and health monitoring.
- **Security**: Distroless images and proper RBAC configuration.
- **Validation**: CEL rules for configuration validation.

For more details about module settings and usage examples, refer to the module documentation.

Module architecture

> **i** The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica per Pod is shown in the diagram.

The level-2 C4 architecture of the `managed-memcached` module and its interactions with other components of DKP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Managed-memcached-operator**: Kubernetes operator consisting of a single container **manager** that performs the following operations:
   - It reconciles Memcached custom resources in all user namespaces. The Memcached resource defines the settings of the Memcached instance, including the topology and the deployment type.
   - It performs Memcached and MemcachedClass custom resources validation, as well as Memcached custom resources mutation using the Validating/Mutating Admission Controllers mechanism.
2. **d8ms-mc-<INSTANCE_NAME>** (StatefulSet): One or more Memcached instances, depending on the deployment type. They are created by the managed-memcached-operator component.

   It consists of a single container:
   - **memcached**: It is an open-source project.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**: Manages Memcached custom resources.

The following external components interact with the module:

1. **Kube-apiserver**: Validates Memcached and MemcachedClass custom resources, mutates Memcached custom resources.
2. **Prometheus-main**: Collects Memcached instances metrics.
3. **User applications**: Sends requests to the Memcached instances.

Additional resources

- Module `managed-memcached` documentation

Managed-postgres module

The `managed-postgres` module manages PostgreSQL clusters in Deckhouse Kubernetes Platform (DKP). It allows users to configure and scale PostgreSQL clusters according to their needs, ensuring optimal performance and security.

Main features:

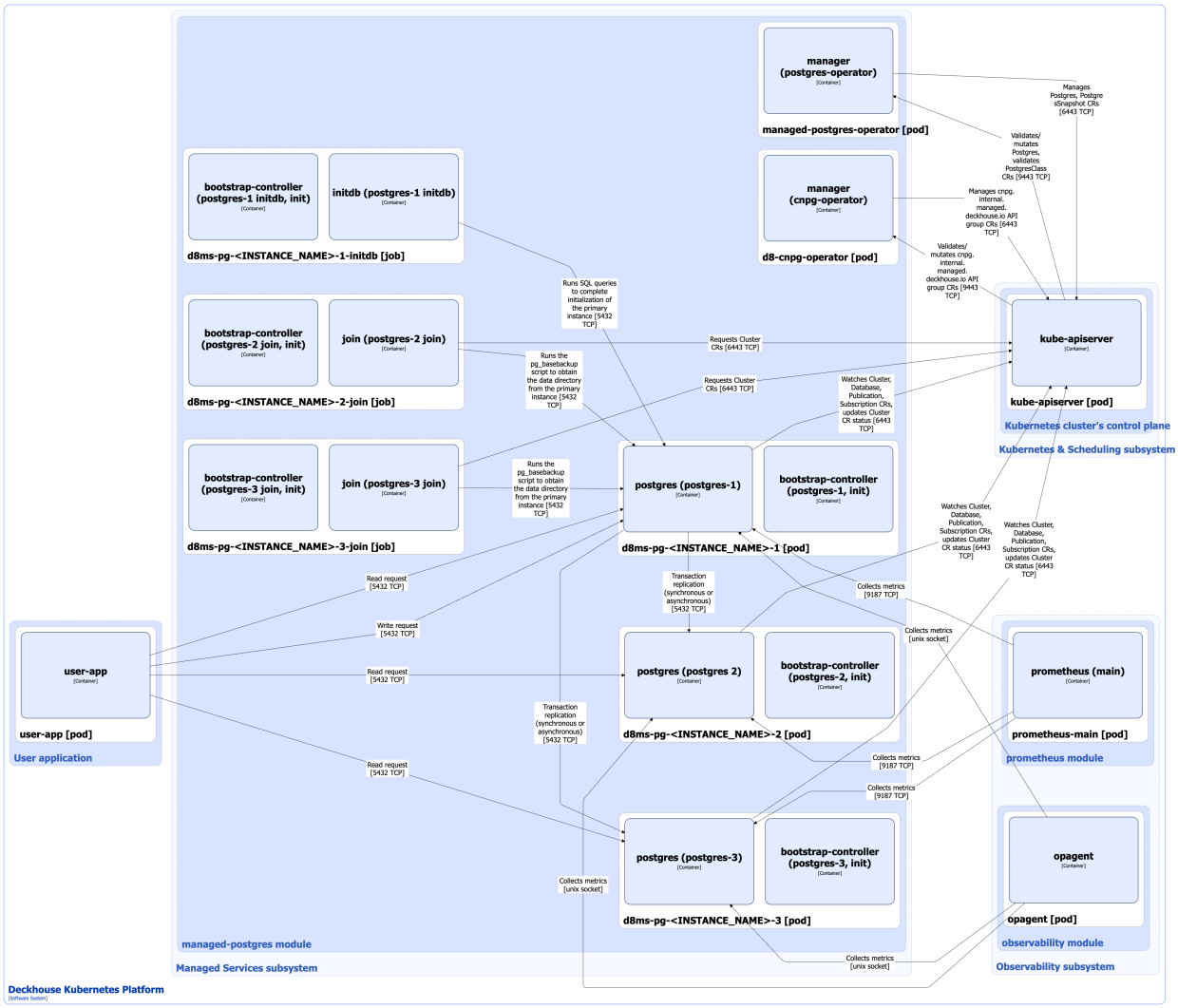
- **Automatic Deployment**: Deploys a Postgres instance using a simple YAML configuration
- **High Availability**: Supports deployment of a highly available Postgres cluster or a standalone instance of your choice.
- **Configuration Management**: Separate PostgresClass custom resource for templating the cluster creation approach with the ability to flexibly validate user configs.
- **User and Database Management**: Declarative model for creating users and logical databases.
- **Status**: Informative set of states for tracking the deployed Postgres.

For more details about module settings and usage examples, refer to the module documentation.

Module architecture

> **i** The following assumptions are used to simplify the diagram:
>
> - The diagram shows direct communication between containers in different Pods. In practice, components communicate through Kubernetes Services (internal load balancers). Service names are omitted when obvious from context. In other cases, a service name is shown above the arrow.
> - Pods can run with multiple replicas, but only one replica per Pod is shown in the diagram.

The level-2 C4 architecture of the `managed-postgres` module and its interactions with other components of DKP are shown in the following diagram:



Module components

The module consists of the following components:

1. **Managed-postgres-operator**: Kubernetes operator consisting of a single container **manager** that performs the following operations:

   - It reconciles Postgres custom resources in all user namespaces. The Postgres resource defines the settings of the PostgreSQL cluster, including the topology and the replication mode, PostgreSQL instances configuration, and other parameters such as lists of logical databases and internal users.
   - It reconciles PostgresSnapshot custom resources. PostgresSnapshot resource designed for PostgreSQL instances backup and restore;
   - It performs Postgres and PostgresClass custom resources validation, as well as Postgres custom resources mutation using the Validating/Mutating Admission Controllers mechanism.

   Managed-postgres-operator uses the `cnpg.internal.managed.deckhouse.io` API group custom resources managed by the d8-cnpg-operator component as a backend.

2. **D8-cnpg-operator**: A fork of CloudNativePG, a Kubernetes operator that automates the management of PostgreSQL clusters. D8-cnpg-operator consists of a single container **manager** and performs the following operations:

   - It manages following `cnpg.internal.managed.deckhouse.io` API group custom resources:
     - Cluster: Defines a PostgreSQL cluster.
     - Pooler: Defines connection pool settings.
     - FailoverQuorum: It is used to display a quorum status of PostgreSQL cluster replicas in High Available configuration.
     - Database: Defines a logical data base.
     - Backup: Defines a PostgreSQL instance backup.
     - ScheduledBackup: Defines a PostgreSQL scheduled backup settings.
     - Subscription: Defines a logical replication destination.
     - Publication: Defines a logical replication source.
     - ImageCatalog: Defines PostgreSQL images for user namespaces.
     - ClusterImageCatalog: Defines cluster-wide PostgreSQL images.
   - It performs `cnpg.internal.managed.deckhouse.io` API group custom resources validation and mutation using the Validating/Mutating Admission Controllers mechanism.

> **i** The components for ConsistencyAndAvailability replication mode are described below. The components for other two replication modes are the part of components for ConsistencyAndAvailability mode.

3. **d8ms-pg-<INSTANCE_NAME>-1-initdb** (Job): A job created by the d8-cnpg-operator component that runs SQL queries to complete initialization of the primary PostgreSQL instance.

   It consists of the following containers:

   - **bootstrap-controller**: Init container that installs `manager` executable file of d8-cnpg-operator component.
   - **initdb**: Main container that runs `manager` executable file, which executes the SQL queries described above.

4. **d8ms-pg-<INSTANCE_NAME>-2-join** (Job): A job created by the d8-cnpg-operator component that runs `pg_basebackup` script to get `data` directory from primary instance for first replica. This job uses a Cluster custom resource to configure the replica.

   It consists of the following containers:

   - **bootstrap-controller**: Init container that installs `manager` executable file of d8-cnpg-operator component.
   - **join**: Main container that runs `manager` executable file, which executes the script described above.

5. **d8ms-pg-<INSTANCE_NAME>-3-join** (Job): A job created by the d8-cnpg-operator component that runs `pg_basebackup` script to get `data` directory from primary instance for second replica. This job uses a Cluster custom resource to configure the replica.

   It consists of the following containers:

   - **bootstrap-controller**: Init container that installs `manager` executable file of d8-cnpg-operator component.
   - **join**: Main container that runs `manager` executable file, which executes the script described above.

6. **d8ms-pg-<INSTANCE_NAME>-1**: PostgreSQL primary instance. It is created by the d8-cnpg-operator component.

   It consists of the following containers:

   - **bootstrap-controller**: Init container that installs `manager` executable file of d8-cnpg-operator component.
   - **postgres**: Main container that runs `manager` executable file, which performs following operations:
     - It starts PostgreSQL processes.
     - It manages PostreSQL instance lifecycle, including server monitoring, handling server shutdown and restart.
     - It participates `switchover`/`failover` procedures. `Switchover` is a planned and controlled process in which an active primary instance is intentionally decommissioned and an assigned backup instance (replica) is promoted to the primary instance role. Its main goal is to ensure zero data loss: before transferring the role to the replica, we wait until all current transactions are replicated. A `failover` is an emergency situation: the primary instance has failed, become unavailable, or cannot be safely used to write data. In this case, the designated backup instance (replica) is promoted to the primary instance role, and data loss is possible.
     - It interacts with the operator and publish instance status;
     - It watches Cluster, Database, Publication and Subscription custom resources.

7. **d8ms-pg-<INSTANCE_NAME>-2**: PostgreSQL first replica. It is created by the d8-cnpg-operator component.

   It consists of the following containers:

   - **bootstrap-controller**: Init container that installs `manager` executable file of d8-cnpg-operator component.
   - **postgres**: Main container that runs `manager` executable file, which performs operations described for PostgreSQL primary instance.

8. **d8ms-pg-<INSTANCE_NAME>-3**: PostgreSQL second replica. It is created by the d8-cnpg-operator component.

   It consists of the following containers:

   - **bootstrap-controller**: Init container that installs `manager` executable file of d8-cnpg-operator component.
   - **postgres**: Main container that runs `manager` executable file, which performs operations described for PostgreSQL primary instance.

Module interactions

The module interacts with the following components:

1. **Kube-apiserver**:
   - Manages Postgres and PostgresSnapshot custom resources.
   - Manages `cnpg.internal.managed.deckhouse.io` API group custom resources.

The following external components interact with the module:

1. **Kube-apiserver**:
   - Validates Postgres and PostgresClass custom resources, mutates Postgres custom resources.
   - Validates and mutates `cnpg.internal.managed.deckhouse.io` API group custom resources.
2. **Prometheus-main**: Collects PostgreSQL cluster instances metrics (which are exported by `manager` executable file of d8-cnpg-operator component that is run in postgres container of PostgreSQL instance pod).
3. **opAgent**: Collects PostgreSQL cluster instances metrics (connecting directly to PostgreSQL cluster instances) and sends them to prometheus-main.

4. **User applications**: Sends requests to PostgreSQL cluster instances. Write requests are sent to the primary instance via the **d8ms-pg-<INSTANCE_NAME>-rw** service. The primary instance replicates transactions to replicas. Read requests are balanced to PostgreSQL instances via **d8ms-pg-<INSTANCE_NAME>-r** or **d8ms-pg-<INSTANCE_NAME>-ro** services.

Additional resources

- Module `managed-postgres` documentation

Virtualization subsystem

This subsection describes the architecture of the Virtualization subsystem of Deckhouse Kubernetes Platform (DKP).

The Virtualization subsystem is represented by the `virtualization` module, which allows you to declaratively create, run, and manage virtual machines (VM) and their resources.

The `virtualization` module consists of the following components:

- Virtualization API (API): A controller that implements a user API for creating and managing VM resources.
- A module core (CORE): It is based on the KubeVirt project and uses QEMU/KVM + libvirtd to run VMs. KubeVirt is an open-source project that allows you to launch, deploy, and manage VMs using Kubernetes as an orchestration platform. It enables a cooperation between traditional VMs and container workloads in the same Kubernetes cluster, providing a single control plane.
- Deckhouse Virtualization Container Registry (DVCR): A container registry for storing and caching VM images.
- Containerized Data Importer (CDI): It is an add-on for managing persistent storage in Kubernetes. Its main goal is to provide a declarative way to create VM disks based on PersistentVolumeClaim (PVC) resources. CDI provides the ability to import VM images and disks into PVC volumes for use in KubeVirt-managed VMs.

- Auxiliary components: Components that implement the following auxiliary functions:
  - Security events audit.
  - Forwarding USB devices to VMs.
  - Updating network routes.
  - Deleting resources before deactivating the `virtualization` module.

For more details about the module, refer to the module documentation section.

Additional resources

- Module `virtualization` documentation

The module core

The `Virtualization` module core is directly responsible for working with virtual machines (VMs). The core is based on the KubeVirt project. KubeVirt is an open-source project that allows you to launch, deploy, and manage VMs using Kubernetes as an orchestration platform. It enables a cooperation between traditional VMs and container workloads in the same Kubernetes cluster, providing a single control plane. A fork of KubeVirt from the company “Flant” is used in the `virtualization` module.

To manage VMs, the module core uses custom resources of the following API groups:

1. `Internal.virtualization.deckhouse.io`: The main group, an analog of the `kubevirt.io` API group of the original KubeVirt. It includes the following custom resources:
   - InternalVirtualizationVirtualMachine: A resource that describes the VM configuration and status.
   - InternalVirtualizationVirtualMachineInstance: A resource that describes a running VM. After VM shutdown, the InternalVirtualizationVirtualMachineInstance resource is deleted, but the InternalVirtualizationVirtualMachine resource that manages the InternalVirtualizationVirtualMachineInstance lifecycle remains.

   The resources of the main group are managed by the virt-controller component. The InternalVirtualizationVirtualMachine resources of the KubeVirt `internal.virtualization.deckhouse.io` main API group are used as the backend for the VirtualMachine resources of `virtualization.deckhouse.io` API group managed by the virtualization-controller.

   > **i** For simplification, the reduced VirtualMachine and VirtualMachineInstance names will be used for the InternalVirtualizationVirtualMachine and InternalVirtualizationVirtualMachineInstance resources (of the original KubeVirt `kubevirt.io` API group) respectively.

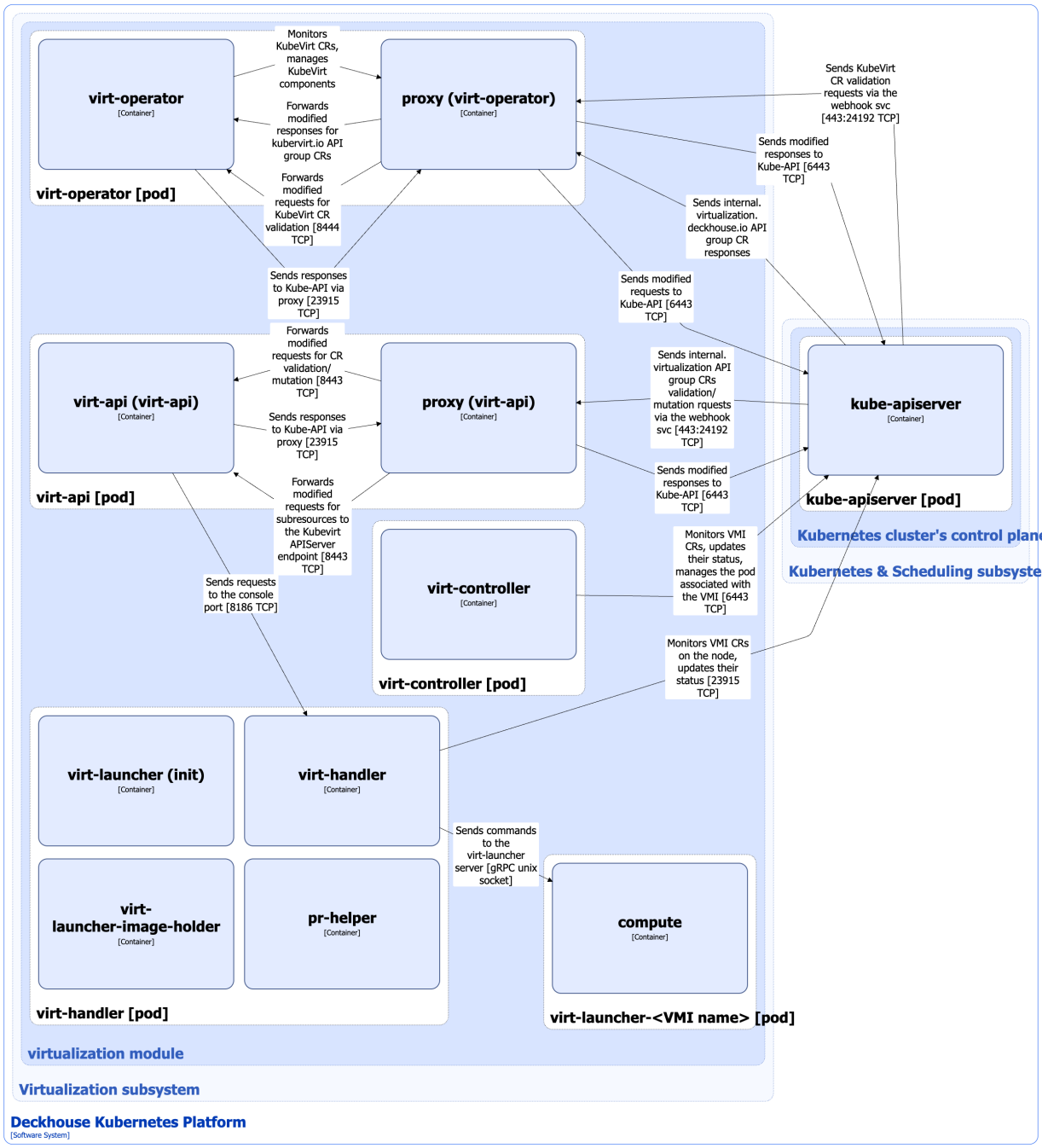
2. `subresources.kubevirt.io`: Subresources group. Subresources are additional operations or actions that can be performed on core resources (for example, VirtualMachineInstance) via the Kubernetes API. They provide interfaces for managing specific aspects of resources without affecting the entire object. Instead of the declarative resource familiar to Kubernetes, they are an endpoint for imperative operations. The following KubeVirt subresources are used in the `virtualization` module:
   - `virtualmachines/{name}/addvolume`;
   - `virtualmachines/{name}/removevolume`;
   - `virtualmachines/{name}/addresourceclaim`;
   - `virtualmachines/{name}/removeresourceclaim`;
   - `virtualmachineinstances/{name}/console`;
   - `virtualmachineinstances/{name}/vnc`;
   - `virtualmachineinstances/{name}/portforward`;
   - `virtualmachineinstances/{name}/freeze`;
   - `virtualmachineinstances/{name}/unfreeze`.

The subresources are managed by the virt-api component. The KubeVirt subresources listed above are used as a backend for similar resources of `subresources.virtualization.deckhouse.io` API group managed by the virtualization-api component.

Module core architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `virtualization` module core and its interactions with other components of DKP are shown in the following diagrams:



Module core components

The module core consists of the following components:

1. **Virt-api**: A Kubernetes Extension API Server that serves requests to the `subresources.kubevirt.io` API groups. The virt-api component performs validation and mutation of custom resources of the `internal.virtualization.deckhouse.io` API groups using the Validating/Mutating Admission Controllers mechanism. Requests pass through the **proxy** sidecar container that renames metadata from the

`internal.virtualization.deckhouse.io` API group to the `kubevirt.io` API group and proxies them to the virt-api component endpoint.

It consists of the following containers:

- **virt-api**: Main container that implements controller and webhook server.
- **proxy** (aka **kube-api-rewriter**): Sidecar container that performs modification of API requests passing through it, namely renaming the metadata of custom resources. This is necessary because KubeVirt components use API groups like `*.kubevirt.io`, and other components of the `virtualization` module use similar resources, but with API groups like `*.virtualization.deckhouse.io`. Kube-api-rewriter is a gateway that proxies requests between controllers that manage resources from different API groups.
- **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the metrics of the proxy container. It is an open-source project.

2. **Virt-controller**: A controller that manages the `internal.virtualization.deckhouse.io` main API group custom resources and responsible for the virtualization functionality at the cluster wide level. For each VirtualMachineInstance resource it creates a separate pod in which the VM is started. Virt-controller monitors VirtualMachineInstance resources, updates their status, and manages the associated pods.

   It consists of the following containers:

   - **virt-controller**: Main container.
   - **proxy** (aka **kube-api-rewriter**): A sidecar container that performs modification of API requests passing through it (described above).
   - **kube-rbac-proxy**: A sidecar container providing authorized access to the metrics of the cdi-apiserver and proxy containers (described above).

3. **Virt-handler** (DaemonSet): A separate controller that runs on all nodes of the cluster. Virt-handler performs the following functions:

   - It extends the kubelet functionality, configuring the pod environment to run a VM inside it. At the moment virt-handler is responsible for creating network interfaces, also it is used to forward `dev/kvm` and other devices from the node into the pod. Virt-handler uses kubelet device plugins to forward devices.

   - Like virt-controller, virt-handler watches VirtualMachineInstance resources corresponding to VMs running on the node. When changes are detected, virt-handler sends a command to the virt-launcher process running in the compute container of the VM pod. Virt-launcher changes the VM state in accordance with the command received. Virt-handler also monitors VM events that virt-launcher returns and synchronizes the corresponding VirtualMachineInstance resource status.

   - It accepts commands from the virt-api component via the `console-port` that correspond to requests for subresources and forwards them to the virt-launcher for execution. By the subresources functionality ports are forwarded to the VM, as well as to the regular and VNC consoles.

     Virt-handler interacts with virt-launcher via the gRPC protocol and Unix socket.

   It consists of the following containers:

   - **virt-launcher**: Init container that runs `node-labeller.sh` script via virt-launcher binary. This script prepares CPU parameters, their functions, and machine types which virt-handler will use to set the appropriate labels on Node resources. These labels, in turn, will be used to schedule VMs to the nodes that support the appropriate parameters.
   - **virt-handler**: Main container.
   - **virt-launcher-image-holder**: A service sidecar container for virt-launcher image pre-loading. The container is on pause and performs only the function of storing the image.
   - **pr-helper**: QEMU persistent reservation helper is a service sidecar container that creates a listener socket that accepts incoming connections to communicate with QEMU. This is necessary because the operating system restricts sending SCSI commands with permanent redundancy to unprivileged programs, which prevents multiple VMs from sharing block SCSI devices, for example in the case of clusterization. QEMU is a free and open-source program for emulating the hardware of various platforms, which is used to run VMs in a pod.

4. **Virt-operator**: A Kubernetes operator that manages the KubeVirt components lifecycle using InternalVirtualizationKubeVirt custom resource. Virt-operator installs the virt-api, virt-controller and virt-handler in the cluster and also configures them.

   It consists of the following containers:

   - **virt-operator**: Main container.
   - **proxy** (aka **kube-api-rewriter**): Sidecar container that performs modification of API requests passing through it (described above).

5. **Virt-launcher-[VMI name]**: The pod where the VM is running (more exactly VirtualMachineInstance).

   It consists of one container:

   - **compute**: A container where the virt-launcher is run. Virt-launcher implements cmd-server (a gRPC server for the remote command execution).

     Depending on the command received from virt-handler, virt-launcher generates an XML specification of the VM being launched or updated and sends it to libvirtd. Libvirtd is a daemon of the server part of the libvirt virtualization management system. It runs on host servers and performs management tasks for virtual guest systems.

     Libvirtd, in turn, starts the VM and manages its lifecycle. The VM is started using QEMU and KVM. QEMU is an open-source emulator that supports hardware virtualization and works in conjunction with the KVM hypervisor.

     In fact libvirtd starts the QEMU process, which is the VM (more exactly VirtualMachineInstance).

Virt-handler also constantly monitors the status of the running VM, returned by libvirtd via virt-launcher, and updates the VirtualMachineInstance status.

Module core interactions

The module core (CORE) interacts with the following components:

1. **Kube-apiserver**:
   - Watches for KubeVirt custom resources and manages KubeVirt components.
   - Watches for VirtualMachineInstance custom resources, updates their status, and manages the associated pods.
   - Authorizes requests for metrics.
2. **CDI (Containerized-Data-Importer)**: KubeVirt creates a DataVolume resource based on the disk specification and a link to the VM image in the `DataVolumeTemplate` section of the VirtualMachine resource. CDI imports a disk image to PVC from the source specified in the DataVolume. The created PVC is a disk of a virtual machine managed by KubeVirt.

The following external components interact with the module core:

1. **Kube-apiserver**:
   - Sends InternalVirtualizationKubeVirt custom resource validating requests.
   - Sends `internal.virtualization.deckhouse.io` API group resource validating and mutating requests.
2. **Prometheus-main**: Collects module core components metrics.

Additional resources

- Module `virtualization` documentation

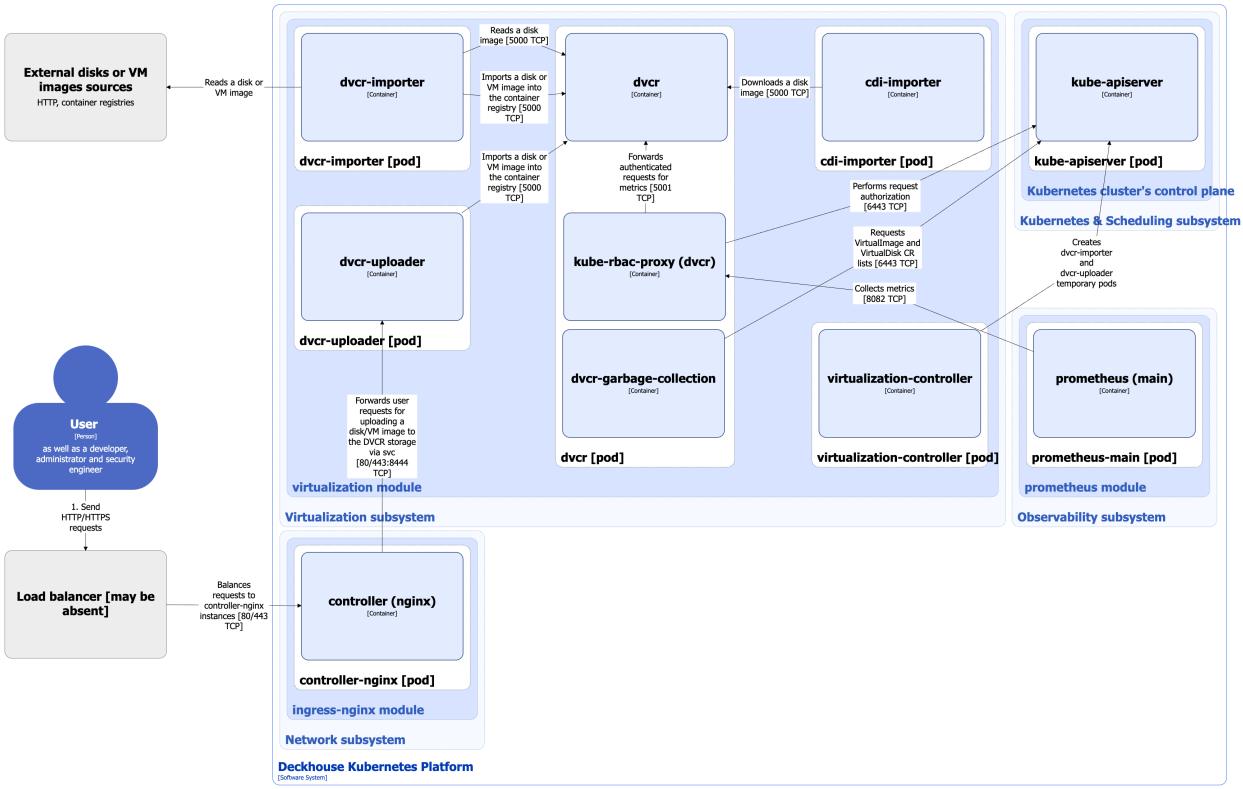
Deckhouse Virtualization Container Registry (DVCR)

The Deckhouse Virtualization Container Registry (DVCR) component of the `virtualization` module is a specialized container registry for storing and caching virtual machine (VM) images. The CDI component of the `virtualization` module uses images stored in DVCR as a source for InternalVirtualizationDataVolume resources, which are used to create disks for KubeVirt-managed VMs.

DVCR architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the DVCR component of `virtualization` module and its interactions with other components of DKP are shown in the following diagrams:



DVCR components

DVCR consists of the following components:

1. **Dvcr**: A container registry based on Distribution. Distribution is an open-source project that provides a framework for storing and distributing container images and other content using the OCI Distribution Specification. Dvcr is used for storing and caching VM images.

   It consists of the following containers:

   - **dvcr**: Main container.
   - **dvcr-garbage-collection**: Sidecar container that periodically deletes images which do not have the appropriate resources in the cluster.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the metrics of the dvcr container. It is an open-source project.

2. **Dvcr-importer**: *Temporary* pod that consists of a single container, run by the virtualization controller to implement various scenarios for importing VM images and disks, such as:

   - Import of a VM disk or image from external sources (HTTP source available via URL or container registry) to the DVCR registry.
   - Import of a VM image from external sources (HTTP source available via URL or container registry) into the PVC volume. Dvcr-importer does not directly import the disk into PVC volume. It uploads the source to the DVCR registry. Next, the InternalVirtualizationDataVolume resource is created, and then CDI imports the image from DVCR storage into PVC volume.
   - Import of a VM image from VirtualImage, ClusterVirtualImage, VirtualDisk or VirtualDiskSnapshot resources to DVCR registry.

3. **Dvcr-uploader**: *Temporary* pod that consists of a single container, run by the virtualization controller to implement following scenarios for user to upload VM images and disks, such as:

   - Upload to DVCR.
   - Upload into PVC volume. Dvcr-uploader does not directly upload the disk into PVC volume. It uploads the source to the DVCR registry. Next, the InternalVirtualizationDataVolume resource is created, and then CDI imports the image from DVCR storage into PVC volume.

DVCR interactions

DVCR interacts with the following components:

1. **Kube-apiserver**: Sends `get`/`list`/`watch`-requests for VirtualImages, ClusterVirtualImages, and VirtualDisks to clean up unused images and for coordination.
2. **External disks or VM images sources**: Reads VM disks or images when implementing some scenarios of import to DVCR storage.

The following external components interact with the DVCR component:

1. **Virtualization-controller**: Starts the dvcr-importer and dvcr-uploader pods to run scripts for VM disks and images import and download.
2. **Ingress-controller**: Forwards user requests to upload a VM disk or image to the DVCR storage via the dvcr-uploader service HTTP endpoint.
3. **Cdi-importer**: Uses images stored in DVCR as a source for InternalVirtualizationDataVolume resources.

Additional resources

- Module `virtualization` documentation

Containerized Data Importer (CDI)

> **i** A fork of CDI is used in the `virtualization` module. The original CDI is a KubeVirt subproject. KubeVirt is an open-source project that allows you to launch, deploy, and manage virtual machines (VMs) using Kubernetes as an orchestration platform.

Containerized Data Importer (CDI) component of the `virtualization` module is a persistent storage management add-on for Kubernetes. It's primary goal is to provide a declarative way to build VM disks based on PersistentVolumeClaim (PVC) resources. CDI provides the ability to import VM images and disks into PVC volumes for use in KubeVirt-managed VM. The data can come from different sources:

- A URL address
- A container registry
- Another PVC (clone)
- A snapshot
- An upload from a client

CDI supports import of two types of data:

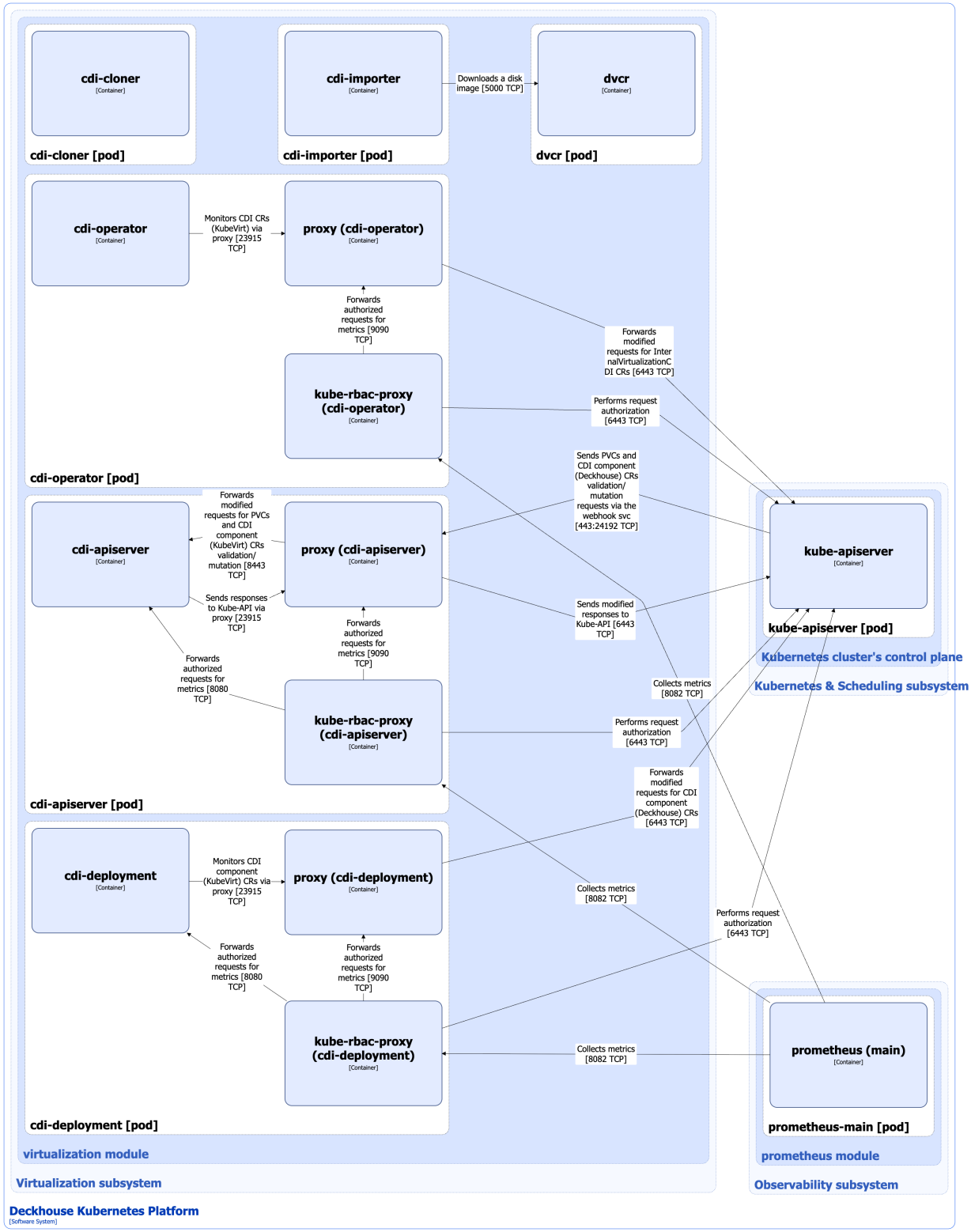
- **KubeVirt data**: Indicates that the file being imported should be treated as a KubeVirt VM disk. CDI will automatically decompress and convert the file from the supported format to `raw` or `qcow2` format (depending on the volume mode). It will also resize the disk to use all available space.
- **Archive data**: Indicates that the data is a TAR archive. Compression is not supported for archives. CDI will extract the contents of the archive into the volume, which can then be used with either a regular pod, or a VM using KubeVirt's filesystem feature.

CDI uses custom resources for disk management. The InternalVirtualizationDataVolume custom resource is an abstraction on top of the standard Kubernetes PVC and can be used to automate creation and population of a PVC with data.

CDI architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the CDI component of `virtualization` module and its interactions with other components of DKP are shown in the following diagrams:



CDI components

CDI consists of the following components:

1. **Cdi-operator**: A Kubernetes operator that manages the CDI components lifecycle using an InternalVirtualizationCDI custom resource. Cdi-operator installs cdi-apiserver and cdi-deployment in the cluster and performs their configuration as well.

   It consists of the following containers:

   - **cdi-operator**: Main container.
   - **proxy** (aka **kube-api-rewriter**): Sidecar container that performs modification of API requests passing through it, namely renaming the metadata of custom resources. This is necessary because KubeVirt components use API groups like `*.kubevirt.io`, and other components of the `virtualization` module use similar resources, but with API groups like `*.virtualization.deckhouse.io`. Kube-api-rewriter is a gateway that proxies requests between controllers that manage resources from different API groups.
   - **kube-rbac-proxy**: Sidecar container with an authorization proxy based on Kubernetes RBAC that provides secure access to the metrics of the proxy container. It is an open-source project.

2. **Cdi-apiserver**: Kubernetes Extension API Server, which is used to validate and mutate Kubernetes API resources through the Validating/Mutating Admission Controllers mechanisms. Cdi-apiserver implements validating and mutating webhooks for the following types of resources:
   - PersistentVolumeClaim: A standard Kubernetes API resource.
   - InternalVirtualizationDataVolume: An abstraction on top of the standard Kubernetes PVC for creating VM disks.
   - InternalVirtualizationCDI: A custom resource used by the cdi-operator to install and configure CDI components.
   - InternalVirtualizationDataImportCron: It defines a cron task for importing disk images as PVCs.
   - VolumeImportSource: It defines the sources for importing disks.

   It consists of the following containers:
   - **cdi-apiserver**: Main container.
   - **proxy** (aka **kube-api-rewriter**): Sidecar container that performs modification of API requests passing through it (described above).
   - **kube-rbac-proxy**: Sidecar container providing authorized access to the metrics of the cdi-apiserver and proxy containers (described above).
3. **Cdi-deployment** (aka **cdi-controller**): A controller that performs the following operations with InternalVirtualizationDataVolumes:
   - Import of VM images and disks into PVC volumes for use in KubeVirt managed VMs.
   - Disk cloning (importing into PVC from other PVC volumes or snapshots).
   - Synchronization of PVC with the corresponding InternalVirtualizationDataVolumes custom resources.

   To perform some of the above operations, the controller creates and launches temporary pods:
   - **cdi-importer**: For importing images and VM disks. Cdi-importer also converts images depending on the type of target PVC:
     - To `raw` format, if the `Block` volume mode is set for the PVC.
     - To `qcow2` format, if the `Filesystem` volume mode is set for the PVC.
   - **cdi-cloner**: For disks and snapshots cloning.

   It consists of the following containers:
   - **cdi-deployment**: Main container, built upon the cdi-controller;
   - **proxy** (aka **kube-api-rewriter**): Described above.
   - **kube-rbac-proxy**: Sidecar container providing authorized access to the metrics of the cdi-deployment and proxy containers (described above).

CDI interactions

CDI interacts with the following components:

1. **Kube-apiserver**:
   - Watches for InternalVirtualizationCDI and PersistentVolumeClaim resources.
   - Authorizes requests for metrics.
2. **KubeVirt**: Provides the ability to populate PVCs with VM images and disks to use them in KubeVirt managed VMs.
3. **DVCR (Deckhouse Virtualization Container Registry)**: Uses the container registry as a source for importing VM images and disks.

The following external components interact with the CDI component:

1. **Kube-apiserver**: Validates/mutates CDI component custom resources as well as PersistentVolumeClaim standard resources.
2. **Prometheus-main**: Collects CDI components metrics.

Additional resources

- Module `virtualization` documentation

Auxiliary components

The `virtualization` module uses components that implement the following auxiliary functions:

- Security events audit.
- Forwarding USB devices to virtual machines (VMs).
- Updating network routes.
- Deleting resources before deactivating the `virtualization` module.

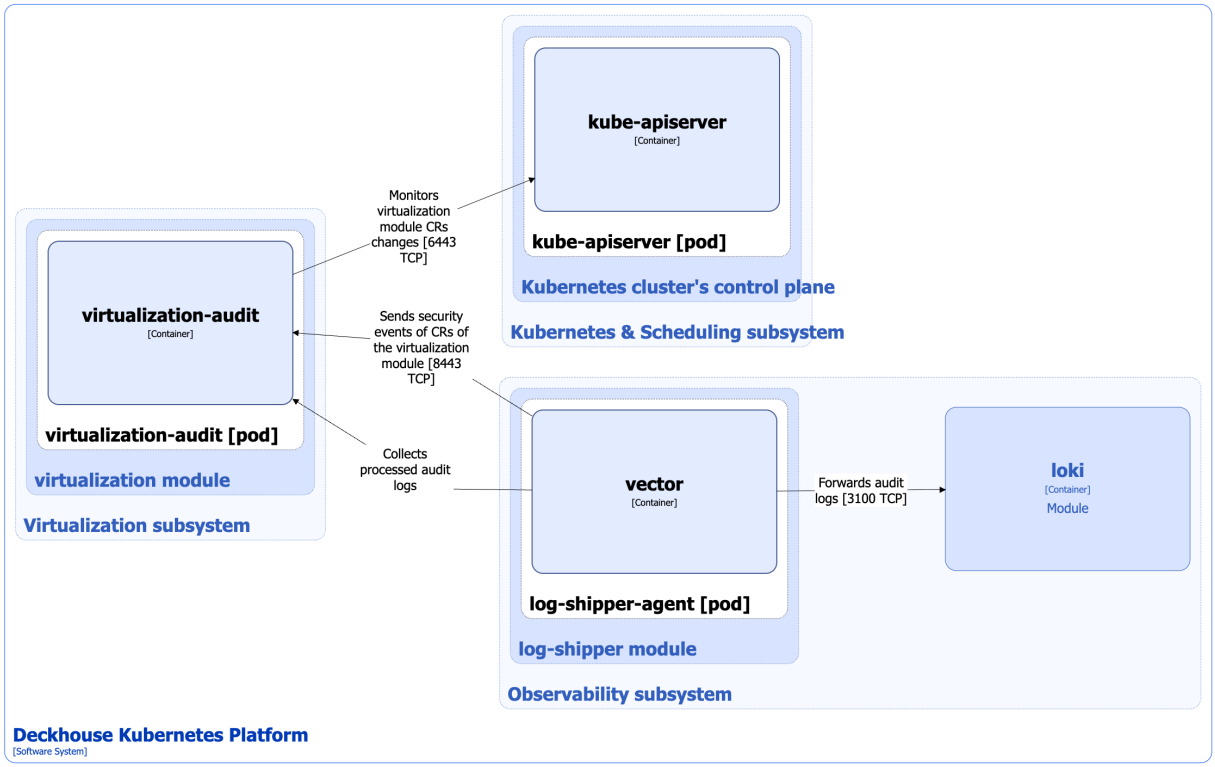
Security events audit

For instructions on `virtualization` module security events audit activation, refer to the module documentation.

Architecture

> **i** The following simplifications are made in the diagram:
>
> - The diagram shows containers in different pods interacting directly with each other. In reality, they communicate via the corresponding Kubernetes Services (internal load balancers). Service names are omitted if they are obvious from the diagram context. Otherwise, the Service name is shown above the arrow.
> - Pods may run multiple replicas. However, each pod is shown as a single replica in the diagram.

The Level 2 C4 architecture of the `virtualization` module auxiliary components for security events audit and their interactions with other components of DKP is shown in the following diagram:



Components

Virtualization-audit is a component consisting of a single container that receives the `virtualization` module security events stream. Sending events is implemented by the `log-shipper` module. The Vector logging agent, according to the settings in the ClusterLoggingConfig custom resources, selects events related to the `virtualization` module custom resources from the cluster's audit log and sends them to the virtualization-audit service endpoint. Virtualization-audit processes the received audit events, enriches them with data from the Kubernetes API, and saves the processed events to its own log.

You can forward security events to the cluster logging system (for example, Loki). In this case, ClusterLoggingConfig resources and the vector agent of the `log-shipper` module are used in a similar way.

Interactions

Virtualization-audit interacts with the following components:

1. **Kube-apiserver**: Watches for `virtualization` module custom resources.

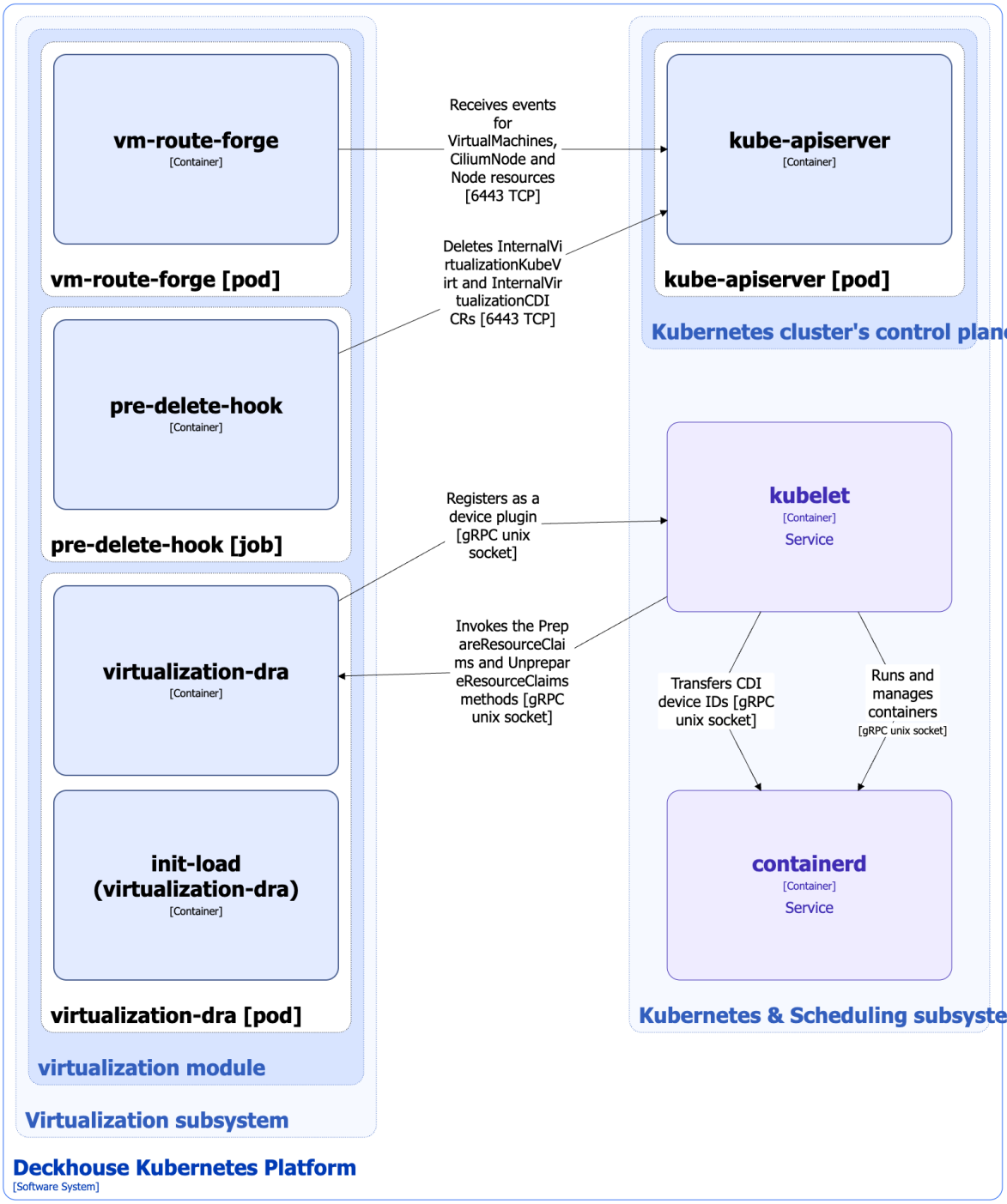
The following external components interact with virtualization-audit:

1. **Log-shipper-agent**:
   - Sends `virtualization` module security events.
   - Collects processed audit logs.

Virtualization-dra and other components

Architecture

The Level 2 C4 architecture of other `virtualization` module auxiliary components and their interactions with other components of DKP is shown in the following diagram:



Components

1. **Virtualization-dra** (DaemonSet): The DRA driver that forwards USB devices to VMs. The DRA (Dynamic Resource Allocation) technology is used to forward USB devices. DRA is a Kubernetes API, scheduler, and kubelet engine for describing, scheduling, and provisioning dynamically allocated resources through external drivers. The DRA driver performs the following operations:

   - Automatically detects USB devices on cluster nodes and publishes them as a ResourceSlice resource. virtualization-controller synchronizes this data into NodeUSBDevice custom resources, which are then used to configure USB device forwarding. For more information about configuring USB device forwarding, refer to the module documentation.
   - Registers in kubelet as the DRA kubelet plugin. The DRA kubelet plugin prepares and releases allocated resources for pods through the PrepareResourceClaims and UnprepareResourceClaims operations. The PrepareResourceClaims method returns CDI (Container Device Interface) device IDs that kubelet passes to containerd. Data on available USB devices is published via ResourceSlice, and device selection is performed by the Kubernetes DRA mechanism based on ResourceClaim/ResourceClaimTemplate and DeviceClass.

     The DRA driver interacts with kubelet via the gRPC protocol and Unix sockets.
   - Implements a USBIP server, so that the USB device is automatically forwarded over the network to the node where the VM is running. There is no need to manually place the VM on the same node where the device is located.

   It consists of the following containers:

    - **init-load**: Init container that loads the Linux kernel modules necessary for the DRA driver operation.
    - **virtualization-dra**: Main container.

2. **Vm-route-forge**: A controller that monitors VirtualMachine custom resources of the `virtualization.deckhouse.io` API group and updates network routes on the node via Linux netlink/eBPF in routing tables used by CNI Cilium for routing traffic between VMs.

3. **Pre-delete-hook** (Job): A Job started by the Deckhouse controller before the `virtualization` module is deleted. It removes the InternalVirtualizationKubeVirt and InternalVirtualizationCDI custom resources named `config`.

Interactions

Virtualization-dra interacts with the following components:

1. **Kubelet**: Registers in kubelet as the DRA kubelet plugin.

Vm-route-forge interacts with the following components:

1. **Kube-apiserver**: Receives events on VirtualMachine, CiliumNode and Node resources.
2. **Host network/Linux kernel**: Updates routes and routing rules on the node.
3. **Cilium data plane**: Uses data from the CiliumNode and Cilium routing tables to route VM traffic.

Pre-delete-hook interacts with the following components:

1. **Kube-apiserver**: Deletes InternalVirtualizationKubeVirt and InternalVirtualizationCDI resources named `config`.

The following external components interact with Virtualization-dra:

1. **Kubelet**: Calls PrepareResourceClaims and UnprepareResourceClaims gRPC methods to prepare and release resources associated with USB devices.

Additional resources

- Module `cni-cilium` documentation
- Module `log-shipper` documentation
- Module `virtualization` documentation

Global configuration

Global configuration settings allow you to customize parameters that are used by default by all modules and components of the Deckhouse Kubernetes Platform. Some modules may override some of these parameters (this can be found in the settings section of the respective module’s documentation).

The global configuration settings are stored in the ModuleConfig `global`.

> **i**
>
> The publicDomainTemplate parameter specifies a DNS name template used by some Deckhouse modules to create Ingress resources. If this parameter is not specified, Ingress resources will not be created.
>
> If you don’t have access to wildcard DNS records, you can use sslip.io or similar services for testing purposes.
>
> The domain specified in the template must not match or be a subdomain of the domain specified in the `clusterDomain` parameter. We do not recommend changing the `clusterDomain` value unless absolutely necessary.
>
> For the template to function correctly, you must first configure DNS services both in the networks where the cluster nodes will be located and in the networks from which clients will access the platform’s service web interfaces.
>
> If the template matches the domain of the node network, use only A records to assign addresses of the nodes’ Frontend interfaces to the platform’s service web interfaces.
> For example, if the nodes are registered under the `company.my` zone and the template is `%s.company.my`.

Example of the `ModuleConfig/global`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: global
spec:
  version: 2
  settings: # <-- Module parameters from the "Parameters" section below.
    defaultClusterStorageClass: 'default-fast'
    modules:
      publicDomainTemplate: '%s.kube.company.my'
      resourcesRequests:
        controlPlane:
          cpu: 1000m
          memory: 500M
      placement:
        customTolerationKeys:
        - dedicated.example.com
      storageClass: 'default-fast'
```

Conversions

Global configuration settings are configured using the `global` ModuleConfig, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Move `storageClass` to `modules.storageClass`. Remove `modules.resourcesRequests.everyNode`, `modules.resourcesRequests.masterNode` and `modules.proxy` parameters.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.defaultClusterStorageClass**
    string

    The name of the StorageClass that DKP will define as the default StorageClass used in the cluster.

    DKP prohibits manually adding or modifying the default annotation (`storageclass.kubernetes.io/is-default-class='true'`) for StorageClass. Use the parameter to define or change the StorageClass that is used as the default in the cluster.

    If the parameter is omitted, the default StorageClass is either:

    - A StorageClass in the cluster that has the default annotation (`storageclass.kubernetes.io/is-default-class='true'`);
    - The first (alphabetically) StorageClass created by the cloud provider module (if any cloud provider module is used).

    Use this parameter to change the default StorageClass in the cluster, as manually adding the default annotation to a StorageClass is not allowed.

    Default:

  - **settings.highAvailability**
    boolean

    A global switch to enable the *high availability* mode for modules that support it.

    If not defined, the value is determined automatically as `true` for clusters with more than one master node. Otherwise, it is determined as `false`.

    Examples:

    ```
    highAvailability: true
    ```

    ```
    highAvailability: false
    ```

  - **settings.modules**
    object

    Common parameters of Deckhouse modules.

    - **settings.modules.gatewayAPIGateway**
      object

      Reference to the Gateway object (from the Gateway API) used by modules of the Deckhouse ecosystem

when creating Gateway API resources.

If not specified, modules use the Gateway automatically discovered in the cluster (if any).

- **settings.modules.gatewayAPIGateway.name**
  string

  Required value

  Name of the Gateway object.

- **settings.modules.gatewayAPIGateway.namespace**
  string

  Required value

  Namespace of the Gateway object.

- **settings.modules.https**
  object

  The HTTPS implementation used by the Deckhouse modules.

  Examples:

```
https:
  certManager:
    clusterIssuerName: letsencrypt
  mode: CertManager
```

```
https:
  mode: Disabled
```

```
https:
  mode: OnlyInURI
```

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: plainstring
```

  - **settings.modules.https.certManager**
    object
    - **settings.modules.https.certManager.clusterIssuerName**
      string

      Name of a `ClusterIssuer` to use for Deckhouse modules.

      The `cert-manager` module offers the following ClusterIssuer options:

      - `letsencrypt`
      - `letsencrypt-staging`
      - `selfsigned`
      - `clouddns`
      - `cloudflare`
      - `digitalocean`
      - `route53`

      You can use your own ClusterIssuer if necessary.

      Default: `letsencrypt`

  - **settings.modules.https.customCertificate**
    object
    - **settings.modules.https.customCertificate.secretName**
      string

      The name of the secret in the `d8-system` namespace to use with Deckhouse modules.

      This secret must have the kubernetes.io/tls format.

      Default: `false`

  - **settings.modules.https.mode**
    string

    The HTTPS usage mode:

    - `CertManager` — Deckhouse modules use HTTPS and get a certificate from the ClusterIssuer defined in the `certManager.clusterIssuerName` parameter;
    - `CustomCertificate` — Deckhouse modules use HTTPS using the certificate from the `d8-system` namespace;
    - `Disabled` — Deckhouse modules use HTTP only (some modules may not work, e.g., user-authn);

    - `OnlyInURI` — Deckhouse modules use HTTP in the expectation that an HTTPS load balancer runs in front of them and terminates HTTPS. Load balancer should provide a redirect from HTTP to HTTPS.

  Default: `CertManager`

  Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.modules.ingressClass**
  string

  Default IngressClass name used by the Deckhouse ecosystem modules when creating Ingress resources.

  Some modules support their own `ingressClass` parameter, which lets you use a different IngressClass instead of the default one.

  Default: `nginx`

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  ingressClass: nginx
  ```

- **settings.modules.placement**
  object

  Parameters regulating the layout of Deckhouse module components.

  - **settings.modules.placement.customTolerationKeys**
    array of strings

    A list of custom toleration keys; use them to allow the deployment of some critical add-ons (such as cni and csi) on dedicated nodes.

    Example:

    ```
    customTolerationKeys:
    - dedicated.example.com
    - node-dedicated.example.com/master
    ```

- **settings.modules.publicDomainTemplate**
  string

  A DNS name template with the `%s` key used as a dynamic part of the string.

  Deckhouse modules use this template when creating Ingress resources.

  For example, given the template `%s.kube.company.my`, the `prometheus` module will create an Ingress resource at the address `grafana.kube.company.my` to provide access to Grafana.

  Pay attention to the following:

  - If this parameter is not specified, Ingress resources will not be created.
  - The domain specified in the template must not match or be a subdomain of the domain specified in the `clusterDomain` parameter. It is not recommended to change the `clusterDomain` value unless absolutely necessary.
  - For the template to function correctly, you must first configure DNS services both in the networks where the cluster nodes will be located and in the networks from which clients will access the platform's service web interfaces.
  - If the template matches the domain of the node network, use only A records to assign addresses of the nodes' Frontend interfaces to the platform's service web interfaces. For example, if the nodes are registered under the `company.my` zone and the template is `%s.company.my`.

  Pattern: `^(%s([-a-z0-9]*[a-z0-9])?|[a-z0-9]([-a-z0-9]*)?%s([-a-z0-9]*)?[a-z0-9]|[a-z0-9]([-a-z0-9]*)?%s)(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Examples:

  ```
  publicDomainTemplate: "%s.kube.company.my"
  ```

  ```
  publicDomainTemplate: kube-%s.company.my
  ```

- **settings.modules.resourcesRequests**
  object

  The amount of resources (CPU and memory) allocated to some components.

  More about resource units in Kubernetes.

  - **settings.modules.resourcesRequests.controlPlane**
    object

    Amount of resources (CPU and memory) allocated to control plane components on each master node.

These settings do not apply if the cluster control plane is managed by a cloud provider (for example, Google Kubernetes Engine or Azure Kubernetes Service).

Example:

```
controlPlane:
  cpu: 1000m
  memory: 500M
```

- **settings.modules.resourcesRequests.controlPlane.cpu**
  string or number

  The combined CPU requests for control-plane components on each master node.

  Pattern: `^[0-9]+m?$`

- **settings.modules.resourcesRequests.controlPlane.memory**
  string

  The combined memory requests for control-plane components on each master node.

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.modules.storageClass**
  string

  The StorageClass to use with all Deckhouse Kubernetes Platform components (Prometheus, Grafana, OpenVPN, etc.).

  - If the value is not specified, the default StorageClass set in the cluster is used.
  - Use this parameter only in exceptional circumstances.
  - This parameter is applied during module activation.

  Default:

- **settings.storageClass**
  Deprecated
  string

  > **Deprecated**. Use settings.modules.storageClass instead.

  The StorageClass to use with all Deckhouse Kubernetes Platform components (Prometheus, Grafana, OpenVPN, etc.).

  - If the value is not specified, the default StorageClass set in the cluster is used.
  - Use this parameter only in exceptional circumstances.
  - This parameter is applied during module activation.

  Default:

Additional resources

- Global custom resources

Custom Resources

Application

Scope: Namespaced
Version: v1alpha1

Application represents a namespace-scoped application instance.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info in the Kubernetes documentation.

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info in the Kubernetes documentation.

- **metadata**
  object
- **spec**
  object

  Required value

  Defines the application configuration.

- **spec.packageName**
  string

  Required value

  Name of the application package to install.

- **spec.packageRepositoryName**
  string

  Name of the repository where the package is located. If not specified, the default repository is used.

- **spec.packageVersion**
  string

  Required value

  Version of the application package to install.

- **spec.releaseChannel**
  string

  Release channel for the application package.

- **spec.settings**
  object

  Configuration settings for the application.

- **status**
  object

  Application status.

  - **status.conditions**
    array of objects

    Conditions reflecting the latest observations of the application state.

    Condition describing one aspect of the current application state.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

    - **status.conditions.message**
      string

      Required value

      message is a human readable message indicating details about the transition. This may be an empty string.

      Maximum length: `32768`

    - **status.conditions.observedGeneration**
      integer

      observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

      Allowed values: `0 <= X`

    - **status.conditions.reason**
      string

      Required value

      reason contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

      Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

      Length: `1..1024`

    - **status.conditions.status**
      string

      Required value

      status of the condition, one of True, False, Unknown.

      Allowed values: `True`, `False`, `Unknown`

        - **status.conditions.type**
          string

          Required value

          type of condition in CamelCase or in foo.example.com/CamelCase.

          Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

          Maximum length: `316`

    - **status.currentVersion**
      object

      Information about the currently installed version.

        - **status.currentVersion.channel**
          string

          Release channel from which the version was installed.

        - **status.currentVersion.version**
          string

          Semantic version of the installed application.

    - **status.lastAppliedConfiguration**
      object

      LastAppliedConfiguration is the effective settings (user configuration merged with config-schema defaults) that drove the most recent successful apply.

    - **status.summary**
      object

      Summary aggregates the high-level user-facing state, message and resolution hint for the application. The controller always populates it on reconcile — every application maps to exactly one lifecycle state — so it is the single source of truth for the UI; clients should not re-derive these values from the conditions. The pointer leaves it absent only before the first status computation.

        - **status.summary.message**
          string

          Message is a human-readable description of the current state.

        - **status.summary.state**
          string

          State is the high-level lifecycle state observed for the application. Always one of: Pending, Failed, Updating, Ready, Degraded, Suspended.

        - **status.summary.tip**
          string

          Tip is a human-readable instruction on how to resolve the current state. Empty when no action is required.

    - **status.tracking**
      object

      Nelm tracking.

ApplicationPackage

Scope: Cluster
Version: v1alpha1

ApplicationPackage represents information about available application package.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info in the Kubernetes documentation

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info in the Kubernetes documentation

- **metadata**
  object
- **status**
  object

  Application package status.

- **status.availableRepositories**
  array of strings

  List of repository names where this application package is available.

- **status.usedBy**
  array of objects

  Information about applications using this package.

  - **status.usedBy.name**
    string

    Name of the application instance.

  - **status.usedBy.namespace**
    string

    Namespace where the application is installed.

  - **status.usedBy.version**
    string

    Version of the package used by this application.

- **status.usedByCount**
  integer

  Number of applications using this package.

ApplicationPackageVersion

Short names: apv

Scope: Cluster
Version: v1alpha1

ApplicationPackageVersion represents a version of an application package.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info in the Kubernetes documentation

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info in the Kubernetes documentation

- **metadata**
  object
- **spec**
  object

  Defines the application package version parameters.

  - **spec.packageName**
    string

    Name of the application package.

  - **spec.packageRepositoryName**
    string

    Name of the package repository containing the package.

  - **spec.packageVersion**
    string

    Version of the application package.

- **status**
  object

  Application package version status.

  - **status.conditions**
    array of objects

    Conditions reflecting the latest observations of the package version state.

    Condition describing one aspect of the current package version state.

    - **status.conditions.lastTransitionTime**
      string

      Required value

lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

- **status.conditions.message**
  string

  Required value

  message is a human readable message indicating details about the transition. This may be an empty string.

  Maximum length: `32768`

- **status.conditions.observedGeneration**
  integer

  observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

  Allowed values: `0 <= X`

- **status.conditions.reason**
  string

  Required value

  reason contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

  Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

  Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  status of the condition, one of True, False, Unknown.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  type of condition in CamelCase or in foo.example.com/CamelCase.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.packageMetadata**
  object

  Metadata about the package such as description, requirements, etc.

  - **status.packageMetadata.category**
    string

    The category this package belongs to.

  - **status.packageMetadata.changelog**
    object

    Information about changes in this version.

    - **status.packageMetadata.changelog.features**
      array of strings

      List of new features in this version.

    - **status.packageMetadata.changelog.fixes**
      array of strings

      List of bug fixes in this version.

  - **status.packageMetadata.description**
    object

    Localized descriptions of the package.

    - **status.packageMetadata.description.en**
      string

      English description of the package.

- **status.packageMetadata.description.ru**
  string

  Russian description of the package.

- **status.packageMetadata.licensing**
  object

  Licensing information for different editions.

  - **status.packageMetadata.licensing.editions**
    object

    Licensing information for different package editions.

    - **status.packageMetadata.licensing.editions.<KEY_NAME>**
      object

      `<KEY_NAME>` — item (key) name.

      - **status.packageMetadata.licensing.editions.<KEY_NAME>.available**
        boolean

        Whether this edition is available for use.

- **status.packageMetadata.requirements**
  object

  The system requirements for this package.

  - **status.packageMetadata.requirements.deckhouse**
    object

    Required Deckhouse Kubernetes Platform version.

    - **status.packageMetadata.requirements.deckhouse.constraint**
      string

      Semver constraint expression.

  - **status.packageMetadata.requirements.kubernetes**
    object

    Required Kubernetes version.

    - **status.packageMetadata.requirements.kubernetes.constraint**
      string

      Semver constraint expression.

  - **status.packageMetadata.requirements.modules**
    object

    Required modules, partitioned into mandatory, conditional, and anyOf dependencies.

    - **status.packageMetadata.requirements.modules.anyOf**
      array of objects

      AnyOf groups of alternative dependencies — at least one member of each group must be installed (and satisfy its constraint, if any) for the package to start. Groups are checker-only and add no edges to the dependency graph.

      PackageModuleGroup is a group of alternative module dependencies. At least one member must be installed (and satisfy its constraint, if any) for the package to start. The Name is required and surfaces in scheduler diagnostics; the Description is optional human-facing documentation.

      - **status.packageMetadata.requirements.modules.anyOf.description**
        string

        Human-readable description of the group’s purpose.

      - **status.packageMetadata.requirements.modules.anyOf.modules**
        array of objects

        Required value

        Alternative module dependencies in this group.

        PackageModuleDependency is a single named module dependency with an optional semver constraint. An empty constraint means “any version”.

        - **status.packageMetadata.requirements.modules.anyOf.modules.constraint**
          string

          Semver constraint expression.

        - **status.packageMetadata.requirements.modules.anyOf.modules.name**
          string

          Required value

Module name.

- **status.packageMetadata.requirements.modules.anyOf.name**
  string

  Required value

  Stable identifier used by the scheduler in diagnostics.

- **status.packageMetadata.requirements.modules.conditional**
  array of objects

  Conditional dependencies — not required to be present, but if installed must satisfy the constraint for the package to function correctly. Replaces the legacy “!optional” suffix from the v1 requirements format.

  PackageModuleDependency is a single named module dependency with an optional semver constraint. An empty constraint means “any version”.

  - **status.packageMetadata.requirements.modules.conditional.constraint**
    string

    Semver constraint expression.

  - **status.packageMetadata.requirements.modules.conditional.name**
    string

    Required value

    Module name.

- **status.packageMetadata.requirements.modules.mandatory**
  array of objects

  Mandatory dependencies — must be present (and satisfy the constraint, if any) for the package to start.

  PackageModuleDependency is a single named module dependency with an optional semver constraint. An empty constraint means “any version”.

  - **status.packageMetadata.requirements.modules.mandatory.constraint**
    string

    Semver constraint expression.

  - **status.packageMetadata.requirements.modules.mandatory.name**
    string

    Required value

    Module name.

- **status.packageMetadata.requirements.modules.noneOf**
  array of objects

  NoneOf groups of forbidden dependencies — no member of any group may be installed for the package to start. A member with no constraint is forbidden at any version; a member with a constraint is forbidden only at versions matching that constraint. Groups are checker-only and add no edges to the dependency graph.

  PackageModuleGroup is a group of alternative module dependencies. At least one member must be installed (and satisfy its constraint, if any) for the package to start. The Name is required and surfaces in scheduler diagnostics; the Description is optional human-facing documentation.

  - **status.packageMetadata.requirements.modules.noneOf.description**
    string

    Human-readable description of the group’s purpose.

  - **status.packageMetadata.requirements.modules.noneOf.modules**
    array of objects

    Required value

    Alternative module dependencies in this group.

    PackageModuleDependency is a single named module dependency with an optional semver constraint. An empty constraint means “any version”.

    - **status.packageMetadata.requirements.modules.noneOf.modules.constraint**
      string

      Semver constraint expression.

    - **status.packageMetadata.requirements.modules.noneOf.modules.name**
      string

      Required value

      Module name.

- **status.packageMetadata.requirements.modules.noneOf.name**
  string

  Required value

  Stable identifier used by the scheduler in diagnostics.

- **status.packageMetadata.stage**
  string

  The development stage of the package (e.g., alpha, beta, stable).

- **status.packageMetadata.versionCompatibilityRules**
  object

  Version compatibility rules for upgrades and downgrades.

  - **status.packageMetadata.versionCompatibilityRules.downgrade**
    object

    Compatibility rules for downgrading from this version.

    - **status.packageMetadata.versionCompatibilityRules.downgrade.allowSkipMajor**
      integer

      How many major versions can be skipped.

    - **status.packageMetadata.versionCompatibilityRules.downgrade.allowSkipMinor**
      integer

      How many minor versions can be skipped.

    - **status.packageMetadata.versionCompatibilityRules.downgrade.allowSkipPatches**
      integer

      How many patch versions can be skipped.

    - **status.packageMetadata.versionCompatibilityRules.downgrade.from**
      string

      Starting version range for compatibility.

    - **status.packageMetadata.versionCompatibilityRules.downgrade.maxRollback**
      integer

      Maximum number of versions that can be rolled back.

    - **status.packageMetadata.versionCompatibilityRules.downgrade.to**
      string

      Ending version range for compatibility.

  - **status.packageMetadata.versionCompatibilityRules.upgrade**
    object

    Compatibility rules for upgrading to this version.

    - **status.packageMetadata.versionCompatibilityRules.upgrade.allowSkipMajor**
      integer

      How many major versions can be skipped.

    - **status.packageMetadata.versionCompatibilityRules.upgrade.allowSkipMinor**
      integer

      How many minor versions can be skipped.

    - **status.packageMetadata.versionCompatibilityRules.upgrade.allowSkipPatches**
      integer

      How many patch versions can be skipped.

    - **status.packageMetadata.versionCompatibilityRules.upgrade.from**
      string

      Starting version range for compatibility.

    - **status.packageMetadata.versionCompatibilityRules.upgrade.maxRollback**
      integer

      Maximum number of versions that can be rolled back.

    - **status.packageMetadata.versionCompatibilityRules.upgrade.to**
      string

      Ending version range for compatibility.

- **status.packageSchemas**
  object

  Schemas for validating settings and values passed to the package.

        - **status.packageSchemas.settingsSchema**
          object

          SettingsSchema is the OpenAPI v3 schema used to validate the user-supplied settings of the package. Stored as an opaque object because its contents form a recursive JSON schema that cannot be expressed structurally in a CRD; the controller validates this subtree in Go when loading package metadata.

        - **status.packageSchemas.valuesSchema**
          object

          ValuesSchema is the OpenAPI v3 schema used to validate the effective values (defaults merged with settings) passed to the package's hooks and charts. Stored as an opaque object because its contents form a recursive JSON schema that cannot be expressed structurally in a CRD; the controller validates this subtree in Go when loading package metadata.

    - **status.usedBy**
      array of objects

      Information about applications that are using this package version.

        - **status.usedBy.name**
          string

          Name of the application instance.

        - **status.usedBy.namespace**
          string

          Namespace where the application is installed.

    - **status.usedByCount**
      integer

      Number of applications using this package version.

ClusterConfiguration

Version: deckhouse.io/v1

General parameters of a cluster.

Defines, for example, network and CRI parameters, control plane version, etc. Some parameters can be changed after the cluster is bootstrapped, during its operation.

To change the `ClusterConfiguration` resource in a running cluster, run the following command:

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

Example:

```
apiVersion: deckhouse.io/v1
kind: ClusterConfiguration
podSubnetNodeCIDRPrefix: '24'
podSubnetCIDR: 10.244.0.0/16
serviceSubnetCIDR: 192.168.0.0/16
kubernetesVersion: '1.31'
clusterDomain: k8s.internal
clusterType: Cloud
cloud:
  prefix: k8s-dev
  provider: Yandex
proxy:
  httpProxy: https://user:password@proxy.company.my:8443
  httpsProxy: https://user:password@proxy.company.my:8443
  noProxy:
  - company.my
```

- **apiVersion**
  string

  Required value

  Version of the Deckhouse API.

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **cloud**
  object

  Cloud provider-related settings (if the `Cloud` clusterType is used).

    - **cloud.prefix**
      string

A prefix of the objects to be created in the cloud.

Is used, for example, to distinguish objects created for different clusters, to configure routing, etc.

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **cloud.provider**
  string

  Required value

  Cloud provider.

  Allowed values: `OpenStack`, `AWS`, `GCP`, `Yandex`, `vSphere`, `Azure`, `VCD`, `Zvirt`, `Dynamix`, `Huaweicloud`, `DVP`

- **clusterDomain**
  string

  Required value

  Cluster domain (used for local routing).

  Please note: the domain must not match the domain used in the DNS name template in the publicDomainTemplate parameter. For example, you cannot set `cluster Domain: cluster.local` and `publicDomainTemplate: %s.cluster.local` at the same time.

  > **i** If you need to change a parameter in a running cluster, it is recommended to use instructions

  Default: `cluster.local`

- **clusterType**
  string

  Required value

  Type of the cluster infrastructure:

  - `Static` — a cluster on bare metal (physical servers) or virtual machines. In the case of virtual machines, it is assumed that Deckhouse doesn't have access to the API for managing virtual machines (they are managed by the administrator using the usual cloud infrastructure tools);
  - `Cloud` — a cluster deployed on the resources of a cloud infrastructure. This type implies that Deckhouse has access to the cloud infrastructure API for managing virtual machines.

  Allowed values: `Cloud`, `Static`

- **defaultCRI**
  string

  The container runtime type that used on cluster nodes (NodeGroups) by default.

  If the value `NotManaged` is used, then Deckhouse does not manage the container runtime (and doesn't install it). In this case, it is necessary to use images for NodeGroups on which the container runtime is already installed.

  If `ContainerdV2` is set, `CgroupsV2` will be used (providing improved security and resource management). To use `ContainerdV2` as the container runtime, cluster nodes must meet the following requirements:

  - Support for `CgroupsV2`.
  - Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
  - Systemd version `244` or newer.
  - Support for `erofs` kernel module.

  Default: `Containerd`

  Allowed values: `Containerd`, `ContainerdV2`, `NotManaged`

- **encryptionAlgorithm**
  string

  Starting from version **1.31**, kubeadm use the specified asymmetric encryption algorithm when generating keys and certificates for the following control-plane components:

  - `apiserver`
  - `apiserver-kubelet-client`
  - `apiserver-etcd-client`
  - `front-proxy-client`
  - `etcd-server`
  - `etcd-peer`
  - `etcd-healthcheck-client`

  Certificates for the components listed above will be reissued using the selected algorithm and key length.

  > **i** **Warning.** When reissuing certificates, the root certificate (**CA**) is not rotated. The root certificate is created with the selected algorithm only during the initial cluster bootstrap.

Default: `RSA-2048`

Allowed values: `RSA-2048`, `RSA-3072`, `RSA-4096`, `ECDSA-P256`

- **kind**
  string

  Required value

  Allowed values: `ClusterConfiguration`

- **kubernetesVersion**
  string

  Required value

  Kubernetes version (control plane components of the cluster).

  Changing a parameter in a running cluster will automatically update the cluster's control plane version.

  If `Automatic` is specified, then the control plane version is used, which is considered stable at the moment. If the stable version of control plane is less than the maximum version that has ever been installed in the cluster, more than 1 minor version, then the version of the cluster will not be changed. The version may change when the minor version of the Deckhouse release is changed (see a corresponding release message).

  Allowed values: `1.31`, `1.32`, `1.33`, `1.34`, `1.35`, `Automatic`

- **podSubnetCIDR**
  string

  Required value

  Address space of the cluster's Pods.

  > **Warning!** Changing this parameter in a running cluster is **extremely dangerous** and can lead to complete cluster failure. The change is blocked by default. To bypass this protection, use `dhctl config edit cluster-configuration --yes-i-am-sane-and-i-understand-what-i-am-doing` command. However, it is **strongly recommended to recreate the cluster** instead of changing this parameter. See documentation for details.

- **podSubnetNodeCIDRPrefix**
  string

  The prefix size of the Pod network allocated to each node.

  This parameter determines how many IP addresses are available for Pods on each individual node. For example, if `podSubnetCIDR: 10.111.0.0/16` and `podSubnetNodeCIDRPrefix: "24"`, each node will receive a `/24` subnet (e.g., `10.111.0.0/24`, `10.111.1.0/24`, etc.), providing up to 254 IP addresses per node (253 usable for Pods, accounting for network and broadcast addresses).

  Calculation examples:

  - With `/16` network and `/24` prefix: maximum **256 nodes** (2^(24-16) = 256)
  - With `/16` network and `/25` prefix: maximum **512 nodes**, but only **126 IPs per node**
  - Each Pod network IP can support up to 2 Pods (one running + one terminating)

  > **Warning!** Changing this parameter in a running cluster is **extremely dangerous** and can lead to complete cluster failure. The change is blocked by default. To bypass this protection, use `dhctl config edit cluster-configuration --yes-i-am-sane-and-i-understand-what-i-am-doing` command. However, it is **strongly recommended to recreate the cluster** instead of changing this parameter. See documentation for details.

  Default: `24`

- **proxy**
  object

  Available in editions: BE, SE, SE+, EE

  Global proxy setup (mainly for working in air-gapped environments).

  The parameters described in this section will be translated into the environment variables `HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY` for all cluster nodes and Deckhouse components. This will result in HTTP(S) requests (curl, git, registry, etc.) to all resources not listed in the `noProxy` parameter being made through a proxy. Note that the `podSubnetCIDR` and `serviceSubnetCIDR` subnets, as well as the `clusterDomain` domain are added to `noProxy` automatically.

  > **Caution!** To avoid using proxies in requests between pods and services located in the cluster node network, make sure you list all the host subnets in the `noProxy` parameter.

  - **proxy.httpProxy**
    string

Available in editions: BE, SE, SE+, EE

Proxy URL for HTTP requests.

If necessary, specify the proxy server’s username, password, and port.

Pattern: `^https?://([!*'();&=+$,/?%#\[\]0-9a-zA-Z\.\-\_]+(\:[!*'();:@&=+$,/?%#\[\]0-9a-zA-Z\.\-\_]+)?@)?[0-9a-zA-Z\.\-]+(\:[0-9]{1,5})?$`

Examples:

```
httpProxy: http://proxy.company.my
```

```
httpProxy: https://user:password@proxy.company.my:8443
```

```
httpProxy: https://DOMAIN%5Cuser:password@proxy.company.my:8443
```

```
httpProxy: https://user%40domain.local:password@proxy.company.my:8443
```

- **proxy.httpsProxy**
  string

  Available in editions: BE, SE, SE+, EE

  Proxy URL for HTTPS requests.

  If necessary, specify the proxy server’s username, password, and port.

  Pattern: `^https?://([!*'();&=+$,/?%#\[\]0-9a-zA-Z\.\-\_]+(\:[!*'();:@&=+$,/?%#\[\]0-9a-zA-Z\.\-\_]+)?@)?[0-9a-zA-Z\.\-]+(\:[0-9]{1,5})?$`

  Examples:

  ```
  httpsProxy: http://proxy.company.my
  ```

  ```
  httpsProxy: https://user:password@proxy.company.my:8443
  ```

  ```
  httpsProxy: https://DOMAIN%5Cuser:password@proxy.company.my:8443
  ```

  ```
  httpsProxy: https://user%40domain.local:password@proxy.company.my:8443
  ```

- **proxy.noProxy**
  array of strings

  Available in editions: BE, SE, SE+, EE

  List of no proxy IP and domain entries.

  For wildcard domains, use a domain name with a dot prefix, e.g., “.example.com”.

  > **Caution.** If the cluster is supposed to have pods interacting with services located in the cluster node network, then specify the list of subnets that are used on the nodes.

  - **Element of the array**
    string

    Pattern: `^[a-z0-9\-\./]+$`

- **serviceSubnetCIDR**
  string

  Required value

  Address space of the cluster’s services.

  > **Warning!** Changing this parameter in a running cluster is **extremely dangerous** and can lead to complete cluster failure. The change is blocked by default. To bypass this protection, use `dhctl config edit cluster-configuration --yes-i-am-sane-and-i-understand-what-i-am-doing` command. However, it is **strongly recommended to recreate the cluster** instead of changing this parameter. See documentation for details.

CNIMigration

Short names: cnim

Scope: Cluster
Version: v1alpha1

Defines the configuration for CNI migration.

- **spec**
  object

  Defines the desired state of CNI migration.

  - **spec.targetCNI**
    string

    The target CNI to migrate to.

- **status**
  object

  Defines the observed state of CNI migration.

  - **status.conditions**
    array of objects

    Reflect the state of the migration as a whole.

    - **status.conditions.lastTransitionTime**
      string

      Required value

    - **status.conditions.message**
      string

      Required value

    - **status.conditions.reason**
      string

      Required value

    - **status.conditions.status**
      string

      Required value

    - **status.conditions.type**
      string

      Required value

  - **status.currentCNI**
    string

    The detected CNI from which the switch is being made.

  - **status.failedSummary**
    array of objects

    Details about nodes that failed the migration.

    - **status.failedSummary.node**
      string
    - **status.failedSummary.reason**
      string

  - **status.nodesFailed**
    integer

    The number of nodes where an error occurred.

  - **status.nodesSucceeded**
    integer

    The number of nodes that have successfully completed the current phase.

  - **status.nodesTotal**
    integer

    The total number of nodes involved in the migration.

  - **status.phase**
    string

    The current step of the migration process.

CNINodeMigration

Short names: cninm

Scope: Cluster
Version: v1alpha1

Defines the configuration for CNI migration on a specific node.

- **spec**

object

Defines the desired state of CNI migration on a node.

- **status**
  object

  Defines the observed state of CNI migration on a node.

  - **status.conditions**
    array of objects

    A list of conditions for this migration.

    - **status.conditions.lastTransitionTime**
      string

      Required value

    - **status.conditions.message**
      string

      Required value

    - **status.conditions.reason**
      string

      Required value

    - **status.conditions.status**
      string

      Required value

    - **status.conditions.type**
      string

      Required value

  - **status.phase**
    string

    The current phase of the migration on the node.

DeckhouseRelease

Scope: Cluster
Version: v1alpha1

Determines the state and parameters for applying a specific release (version) of the Deckhouse Kubernetes Platform (DKP) in the cluster.

DeckhouseRelease objects are automatically created by DKP upon the discovery of new DKP versions on the selected release channel. Modifying DeckhouseRelease enables the management of the process for applying the corresponding DKP version. More details about configuring DKP updates can be found in the documentation.

The current versions of DKP and modules by release channels can be found at releases.deckhouse.ru.

- **approved**
  boolean

  Allows or disables manual updates.

  Used only if the `deckhouse` module is configured for manual update mode (update.mode parameter is set to `Manual`). Ignored if the update mode is set to `Auto` or `AutoPatch`.

  For more information on confirming manual updates, refer to the documentation.

  Default: `false`

- **spec**
  object

  Required value

  - **spec.applyAfter**
    string

    Marks release as a part of canary-release. This release will be delayed until the specified time. If the release is waiting to be applied, check `.status.message` for the reason.

  - **spec.changelog**
    object

    A structure containing a list of DKP changes (and modules included in the DKP) of the release.

    For more information about DKP release changelogs, refer to Updating.

  - **spec.changelogLink**
    string

Link to site with full changelog for this release.

- **spec.disruptions**
  Deprecated
  array of strings

  Disruptive changes in the release.

- **spec.requirements**
  object

  A structure containing a list of requirements for installing the release. It is used by the DKP core. If the requirements are not met, the release may be skipped or blocked from installation.

  Reports on unmet requirements can be found in the field `.status.message` of the DeckhouseRelease object.

- **spec.version**
  string

  Required value

  DKP version.

  Example:

  ```
  version: v1.73.2
  ```

- **status**
  object
  - **status.approved**
    boolean

    The status of the release's readiness for deployment. It makes sense only for Manual updates (`update.mode: Manual`).

  - **status.message**
    string

    Detailed status or error message.

  - **status.phase**
    string

    Current status of the release.

    Allowed values: `Pending`, `Deployed`, `Outdated`, `Suspended`, `Superseded`, `Skipped`

  - **status.transitionTime**
    string

    Time of release status change.

InitConfiguration

Version: deckhouse.io/v1

Deckhouse configuration to start after installation.

Example:

```
apiVersion: deckhouse.io/v1
kind: InitConfiguration
deckhouse:
  imagesRepo: nexus.company.my/deckhouse/ee
  registryDockerCfg:
eyJhdXRocyI6IHsgIm5leHVzLmNvbXBhbnkubXkiOiB7InVzZXJuYW1lIjoibmV4dXMtdXNlciIsInBhc3N3b3JkIjoibmV4dX
MtcEBzc3cwcmQiLCJhdXRoIjoiYm1WNGRYTXRkWE5sY2pwdVpYaDFjeTF3UUhOemR6QnlaQW89In19fQo=
  registryScheme: HTTPS
  registryCA: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

- **apiVersion**
  string

  Required value

  Version of the Deckhouse API.

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **deckhouse**
  Deprecated
  object

Required value

Initial parameters required to install Deckhouse.

- **deckhouse.devBranch**
  Deprecated
  string

  The parameter is used for development needs. Will be replaced with the CLI-tools.

- **deckhouse.imagesRepo**
  string

  Address of a container registry with Deckhouse images.

  Specify it if Deckhouse Enterprise Edition edition or third-party registry (e.g. proxy server in a closed environment) is used.

  The address matches the edition of Deckhouse used. The public container registry address for Deckhouse Enterprise Edition is `registry.deckhouse.io/deckhouse/ee`.

  Default: `registry.deckhouse.io/deckhouse/ce`

  Pattern: `^[0-9a-zA-Z\.\-]+(\:[0-9]{1,5})?(\/[0-9a-zA-Z\.\-\_\/]+)?$`

- **deckhouse.registryCA**
  string

  Root CA certificate to validate the container registry's HTTPS certificate (if self-signed certificates are used).

- **deckhouse.registryDockerCfg**
  string

  A Base64-encoded string from the Docker client configuration file (in Linux it is usually `$HOME/.docker/config.json`), for accessing a third-party container registry.

  For example, to access the container registry `registry.company.my` under the user `user` with the password `P@ssw0rd` it will be `eyJhdXRocyI6eyJyZWdpc3RyeS5jb21wYW55Lm15Ijp7ImF1dGgiOiJkWE5sY2pwUVFITnpkekJ5WkFvPSJ9fX0K` (string `{"auths":{"registry.company.my":{"auth":"dXNlcjpQQHNzdzByZAo="}}}` in Base64).

  Default:

  Example:

  ```
  registryDockerCfg: eyJhdXRocyI6IHsgInJlZ2lzdHJ5LmRlY2tob3VzZS5pbyI6IHt9fX0=
  ```

- **deckhouse.registryScheme**
  string

  Registry access scheme (HTTP or HTTPS).

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **kind**
  string

  Required value

  Allowed values: `InitConfiguration`

Module

Scope: Cluster
Version: v1alpha1

Describes the module's status in the cluster. The `Module` object is created automatically after configuring the ModuleSource and successfully completing synchronization.

- **properties**
  object
  - **properties.accessibility**
    object

    Module accessibility settings.

    - **properties.accessibility.editions**
      object

      Module operation settings in Deckhouse editions.

      - **properties.accessibility.editions.<KEY_NAME>**
        object

        `<KEY_NAME>` — item (key) name.

        - **properties.accessibility.editions.<KEY_NAME>.available**
          boolean

        - **properties.accessibility.editions.<KEY_NAME>.enabledInBundles**
          array of strings
    - **properties.availableSources**
      array of strings

      Available sources for downloading the module.

    - **properties.critical**
      boolean

      Indicates whether the module critical or not.

    - **properties.disableOptions**
      object

      Parameters of module disable protection.

        - **properties.disableOptions.confirmation**
          boolean
        - **properties.disableOptions.message**
          string
    - **properties.exclusiveGroup**
      string

      Indicates the group where only one module can be active at a time.

    - **properties.namespace**
      string

      Module namespace.

    - **properties.releaseChannel**
      string

      Module release channel.

    - **properties.requirements**
      object

      Module dependencies, a set of requirements that must be met for Deckhouse Kubernetes Platform (DKP) to run the module.

        - **properties.requirements.bootstrapped**
          string

          Required cluster installation status (for built-in DKP modules only).

        - **properties.requirements.deckhouse**
          string

          Required Deckhouse version.

        - **properties.requirements.kubernetes**
          string

          Required Kubernetes version.

        - **properties.requirements.modules**
          object

          A list of other enabled modules required for the module.

    - **properties.source**
      string

      Source the module was downloaded from (otherwise will be blank).

    - **properties.stage**
      string

      Current stage of the module lifecycle.

    - **properties.subsystems**
      array of strings

      Module subsystems.

    - **properties.updatePolicy**
      string

      Module update policy.

    - **properties.version**
      string

      Module version.

    - **properties.weight**
      integer

      Module *weight* (priority).

- **status**
  object
  - **status.conditions**
    array of objects
    - **status.conditions.lastProbeTime**
      string
    - **status.conditions.lastTransitionTime**
      string
    - **status.conditions.message**
      string
    - **status.conditions.reason**
      string
    - **status.conditions.status**
      string
    - **status.conditions.type**
      string
  - **status.hooksState**
    string

    Hooks status report.

  - **status.phase**
    string

    Module phase.

    Allowed values: `Unavailable`, `Available`, `Downloading`, `DownloadingError`, `Reconciling`, `Installing`, `HooksDisabled`, `WaitSyncTasks`, `Downloaded`, `Conflict`, `Ready`, `Error`

ModuleConfig

Short names: mc

Scope: Cluster
Version: v1alpha1

Defines the configuration of the Deckhouse Kubernetes Platform module (module parameters). The name of the ModuleConfig resource must match the name of the module (for example, `control-plane-manager` for the `control-plane-manager` module).

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: module-1
spec:
  enabled: true
  settings: {}
  version: 1
```

- **spec**
  object

  Required value

  - **spec.enabled**
    boolean

    Enables or disables the module.

    Example:

    ```
    enabled: false
    ```

  - **spec.maintenance**
    string

    Defines the module maintenance mode.

    - `NoResourceReconciliation`: A mode for developing or tweaking the module.

      In this mode:

      - Configuration or hook changes are not reconciled, which prevents resources from being updated automatically.
      - Resource monitoring is disabled, which prevents deleted resources from being restored.
      - All the module's resources are labeled with `maintenance: NoResourceReconciliation`.
      - The `ModuleIsInMaintenanceMode` alert is triggered.

    Allowed values: `NoResourceReconciliation`

    Example:

```
maintenance: NoResourceReconciliation
```

  - **spec.settings**
    object

    Module settings.

  - **spec.source**
    string

    The source of the module it provided by one (otherwise empty).

  - **spec.updatePolicy**
    string

    Module update policy.

    Example:

    ```
    updatePolicy: test-alpha
    ```

  - **spec.version**
    number

    Version of settings schema.

    Example:

    ```
    version: 1
    ```

- **status**
  object
  - **status.message**
    string

    Additional information

  - **status.version**
    string

    Version of settings schema in use

ModuleDocumentation

Scope: Cluster
Version: v1alpha1

Defines the rendering configuration of the Deckhouse module documentation.

Deckhouse creates ModuleDocumentation resources by itself.

- **spec**
  object

  Required value

  - **spec.checksum**
    string

    Module version checksum.

  - **spec.path**
    string

    Path to the module version.

  - **spec.version**
    string

    Required value

    Module version.

    Example:

    ```
    version: v1.0.0
    ```

- **status**
  object
  - **status.conditions**
    array of objects
    - **status.conditions.address**
      string
    - **status.conditions.checksum**
      string
    - **status.conditions.lastTransitionTime**

string
    - **status.conditions.message**
      string
    - **status.conditions.type**
      string
    - **status.conditions.version**
      string
  - **status.result**
    string

ModulePullOverride

Short names: mpo

Scope: Cluster

- v1alpha2
- v1alpha1

Defines the resource configuration for downloading specific versions of Deckhouse modules.

> **i** **Caution**. This resource is intended for development and debugging environments only. Using it in production clusters is not recommended. Support for the resource might be removed in future Deckhouse Kubernetes Platform versions.

- **spec**
  object

  Required value

  - **spec.imageTag**
    string

    Required value

    Module container image tag, which will be pulled.

  - **spec.rollback**
    boolean

    Indicates whether the module release should be rollback after deleting mpo.

    Default: `false`

  - **spec.scanInterval**
    string

    Scan interval for checking the image digest. If the digest changes, the module is updated. Supported units: `h`, `m`, `s`.

    Default: `15s`

    Pattern: `^(\d+h)?(\d+m)?(\d+s)?$`

- **status**
  object
  - **status.imageDigest**
    string

    Digest of the module image.

  - **status.message**
    string

    Details of the resource status.

  - **status.updatedAt**
    string

    When the module was last updated.

  - **status.weight**
    integer

    Module weight.

Deprecated resource. Support for the resource might be removed in a later release.

Defines the configuration.

- **spec**
  object

  Required value

  - **spec.imageTag**
    string

Required value

Module container image tag, which will be pulled.

- **spec.rollback**
  boolean

  Indicates whether the module release should be rollback after deleting ModulePullOverride.

  Default: `false`

- **spec.scanInterval**
  string

  Scan interval for checking the image digest. If the digest changes, the module is updated. Supported units: `h`, `m`, `s`.

  Default: `15s`

  Pattern: `^(\d+h)?(\d+m)?(\d+s)?$`

- **spec.source**
  string

  Required value

  Reference to the ModuleSource with the module.

- **status**
  object
  - **status.imageDigest**
    string

    Digest of the module image.

  - **status.message**
    string

    Details of the resource status.

  - **status.updatedAt**
    string

    When the module was last updated.

  - **status.weight**
    integer

    Module weight.

ModuleRelease

Short names: mr

Scope: Cluster
Version: v1alpha1

Defines the configuration for a Deckhouse release.

ModuleRelease resources are created by Deckhouse.

- **spec**
  object

  Required value

  - **spec.applyAfter**
    string

    Time until which the release will be delayed.

  - **spec.changelog**
    object

    Release’s changelog for the module.

  - **spec.moduleName**
    string

    Required value

    Module name.

  - **spec.requirements**
    object

    Release dependencies, a set of requirements that must be met for Deckhouse Kubernetes Platform to run the module release.

    - **spec.requirements.deckhouse**
      string

Required Deckhouse version.

- **spec.requirements.kubernetes**
  string

  Required Kubernetes version.

- **spec.requirements.modules**
  object

  A list of other modules required for the module release. Ensure the modules are enabled.

- **spec.update**
  object

  Optional transition rules.

  - **spec.update.versions**
    array of objects

    List of `from` → `to` transition rules that allow skipping step-by-step updates. If the current installed module version (status `Deployed`) is not lower than `from`, and the cluster has a release whose version matches `to`, the controller will skip intermediate releases and update the module to the version from `to`. The `to` value can specify a minor line (`X.Y` — the latest available `X.Y.Z` will be selected). The rule is specified in the constrained release — the one whose version matches `to`.

    - **spec.update.versions.from**
      string

      Required value

      The minimum version from which the transition is allowed (format `X.Y`).

    - **spec.update.versions.to**
      string

      Required value

      The end version of the range — a minor line (`X.Y`).

- **spec.version**
  string

  Required value

  Module version.

  Example:

  ```
  version: v1.0.0
  ```

- **spec.weight**
  integer

  Module *weight* (priority).

- **status**
  object
  - **status.approved**
    boolean

    Status indicating that the release is ready for deployment. For the Manual update mode only (`update.mode: Manual`).

  - **status.message**
    string

    Detailed status or error message.

  - **status.phase**
    string

    Current status of the release.

    Allowed values: `Pending`, `Deployed`, `Superseded`, `Suspended`, `Skipped`, `Terminating`

  - **status.pullDuration**
    string

    Module loading duration.

  - **status.size**
    integer

    Size of the module image.

  - **status.transitionTime**
    string

    Time of release status change.

ModuleSettingsDefinition

Scope: Cluster
Version: v1alpha1

It displays module settings. Defines a list of module settings versions.

- **spec**
  object

  Required value

  Specification of the module settings.

  - **spec.versions**
    array of objects

    List of module settings versions. Each version includes a name and a schema.

    - **spec.versions.conversions**
      array of objects

      List of conversion rules for this version.

      A single conversion rule with expressions and descriptions.

      - **spec.versions.conversions.descriptions**
        object

        Localized descriptions of the conversion.

        - **spec.versions.conversions.descriptions.en**
          string

          English description of the conversion.

        - **spec.versions.conversions.descriptions.ru**
          string

          Russian description of the conversion.

      - **spec.versions.conversions.expr**
        array of strings

        Array of jq expressions to transform settings.

    - **spec.versions.name**
      string

      Required value

      Module settings version.

    - **spec.versions.schema**
      object

      Settings schema for the given module version.

ModuleSource

Short names: ms

Scope: Cluster
Version: v1alpha1

Defines the configuration of a source of Deckhouse modules.

For more information about installing the module from the source, see the section “Running the module in the DKP cluster”.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleSource
metadata:
  name: example
spec:
  registry:
    repo: registry.example.io/modules-source
    dockerCfg: "<base64 encoded credentials>"
```

- **spec**
  object

  Required value

  - **spec.registry**

object

Required value

- **spec.registry.ca**
  string

  Root CA certificate (PEM format) to validate the registry's HTTPS certificate (if self-signed certificates are used).

  > **i** Creating a ModuleSource resource with the CA certificate spec will cause the container to restart on all nodes.

- **spec.registry.dockerCfg**
  string

  Container registry access token in Base64. If using anonymous access to the container registry, do not fill in this field.

- **spec.registry.repo**
  string

  Required value

  URL of the container registry.

  Example:

  ```
  repo: registry.example.io/deckhouse/modules
  ```

- **spec.registry.scheme**
  string

  Protocol to access the registry.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

- **spec.releaseChannel**
  Deprecated
  string

  Desirable default release channel for modules in the current source.

- **spec.scanInterval**
  string

  Interval for registry scan.

  Defines the frequency of checking the container registry for new modules and their versions.

  Default: `3m`

  Pattern: `^(\d+h)?(\d+m)?(\d+s)?$`

  Examples:

  ```
  scanInterval: 5m
  ```

  ```
  scanInterval: 1h
  ```

  ```
  scanInterval: 6h30m
  ```

- **status**
  object
  - **status.message**
    string
  - **status.modules**
    array of objects

    The list of modules available from the source and their update policies.

    - **status.modules.checksum**
      string

      The module checksum.

    - **status.modules.error**
      string

      The module processing error.

    - **status.modules.name**
      string

The module name.

- **status.modules.overridden**
  boolean

  If ModulePullOverride for this module exists.

- **status.modules.policy**
  string

  The module policy name.

- **status.modules.pullError**
  string

  The module pull error.

- **status.modules.version**
  string

  The module version.

- **status.modulesCount**
  integer

  The number of modules available.

- **status.phase**
  string

  The current phase.

  Allowed values: `Active`, `Terminating`

- **status.syncTime**
  string

  When the repository was synchronized.

ModuleUpdatePolicy

Short names: mup

Scope: Cluster

- v1alpha2
- v1alpha1

Defines the update settings for a module's release.

Example:

```
apiVersion: deckhouse.io/v1alpha2
kind: ModuleUpdatePolicy
metadata:
  name: example-update-policy
spec:
  releaseChannel: Alpha
  update:
    mode: Auto
    windows:
    - days:
      - Mon
      - Wed
      from: '13:30'
      to: '14:00'
```

- **spec**
  object

  Required value

  - **spec.releaseChannel**
    string

    Desirable module release channel.

    The order in which the stability of the release channel increases (from **less** stable to **more** stable): `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`.

    Default: `Stable`

    Allowed values: `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`, `Lts`

  - **spec.update**
    object

Required value

Update settings for target modules.

- **spec.update.mode**
  string

  Modules version update mode (release change).

  - `AutoPatch` — automatic update mode for patch releases.

    To change a minor version (for example, from `v1.15.*` to `v1.16.*`), confirmation is required.

    A patch version update (for example, from `v1.16.1` to `v1.16.2`) is applied according to the update windows, if they are set.

  - `Auto` — all updates are applied automatically.

    Modules minor version updates (for example, from `v1.15.*` to `v1.16.*`) and patch version updates (for example, from `v1.16.1` to `v1.16.2`) are applied according to the update windows or (if no update windows are set) as they appear on the corresponding release channel;

  - `Manual` — confirmation is required for updating both minor and patch versions.

  To confirm the update, add the `modules.deckhouse.io/approved="true"` annotation to the corresponding ModuleRelease resource.

  Default: `Auto`

  Allowed values: `Auto`, `Manual`, `AutoPatch`

- **spec.update.windows**
  array of objects

  Modules update timetable.

  - **spec.update.windows.days**
    array of strings

    The days of the week on which the update window is applied.

    Examples:

    ```
    days: Mon
    ```

    ```
    days: Wed
    ```

    - **Element of the array**
      string

      Day of the week.

      Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

      Example:

      ```
      Mon
      ```

  - **spec.update.windows.from**
    string

    Required value

    Start time of the update window (UTC timezone).

    Should be less than the end time of the update window.

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    from: '13:00'
    ```

  - **spec.update.windows.to**
    string

    Required value

    End time of the update window (UTC timezone).

    Should be more than the start time of the update window.

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    to: '18:30'
    ```

Deprecated resource. Support for the resource might be removed in a later release.

Defines the update settings for a module’s release.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleUpdatePolicy
metadata:
  name: example-update-policy
spec:
  moduleReleaseSelector:
    labelSelector:
      matchLabels:
        source: example
        module: module-1
  releaseChannel: Alpha
  update:
    mode: Auto
    windows:
    - days:
      - Mon
      - Wed
      from: '13:30'
      to: '14:00'
```

- **spec**
  object

  Required value

  - **spec.moduleReleaseSelector**
    object

    Required value

    Selects target modules to apply update settings to.

    - **spec.moduleReleaseSelector.labelSelector**
      object

      Required value

      Label-selector-based filter to match target modules.

      If both `matchExpressions` and `matchLabels` parameters are set, their requirements are ANDed together — they must all be satisfied in order to match. If multiple `matchExpression` conditions are provided, they all must be satisfied in order to match.

      - **spec.moduleReleaseSelector.labelSelector.matchExpressions**
        array of objects

        An array of set-based expressions.

        - **spec.moduleReleaseSelector.labelSelector.matchExpressions.key**
          string

          Required value

          A label name.

        - **spec.moduleReleaseSelector.labelSelector.matchExpressions.operator**
          string

          Required value

          A comparison operator.

          Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

        - **spec.moduleReleaseSelector.labelSelector.matchExpressions.values**
          array of strings

          A label value.

      - **spec.moduleReleaseSelector.labelSelector.matchLabels**
        object

        A number of equality-based label filters.

        Example:

```
matchLabels:
  source: deckhouse
  module: deckhouse-admin
```

- **spec.releaseChannel**
  string

  Desirable module release channel.

  The order in which the stability of the release channel increases (from **less** stable to **more** stable): `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`.

  Default: `Stable`

  Allowed values: `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`

- **spec.update**
  object

  Required value

  Update settings for target modules.

  - **spec.update.mode**
    string

    Modules version update mode (release change).

    - `AutoPatch` — automatic update mode for patch releases.

      To change a minor version (for example, from `v1.15.*` to `v1.16.*`), confirmation is required.

      A patch version update (for example, from `v1.16.1` to `v1.16.2`) is applied according to the update windows, if they are set.

    - `Auto` — all updates are applied automatically.

      Modules minor version updates (for example, from `v1.15.*` to `v1.16.*`) and patch version updates (for example, from `v1.16.1` to `v1.16.2`) are applied according to the update windows or (if no update windows are set) as they appear on the corresponding release channel;

    - `Manual` — confirmation is required for updating both minor and patch versions.

    To confirm the update, add the `modules.deckhouse.io/approved="true"` annotation to the corresponding ModuleRelease resource.

    - `Ignore` — updates are ignored.

    Default: `AutoPatch`

    Allowed values: `Auto`, `Manual`, `Ignore`, `AutoPatch`

  - **spec.update.windows**
    array of objects

    Modules update timetable.

    - **spec.update.windows.days**
      array of strings

      The days of the week on which the update window is applied.

      Examples:

      ```
      days: Mon
      ```

      ```
      days: Wed
      ```

      - **Element of the array**
        string

        Day of the week.

        Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

        Example:

        ```
        Mon
        ```

    - **spec.update.windows.from**
      string

      Required value

      Start time of the update window (UTC timezone).

      Should be less than the end time of the update window.

Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

Example:

```
from: '13:00'
```

- **spec.update.windows.to**
  string

  Required value

  End time of the update window (UTC timezone).

  Should be more than the start time of the update window.

  Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

  Example:

```
to: '18:30'
```

PackageRepository

Scope: Cluster
Version: v1alpha1

PackageRepository is a source of packages for Deckhouse Kubernetes Platform.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info in the Kubernetes documentation.

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info in the Kubernetes documentation.

- **metadata**
  object
- **spec**
  object

  Required value

  Defines the package repository configuration.

  - **spec.registry**
    object

    Required value

    Configuration for accessing the container registry with packages.

    - **spec.registry.ca**
      string

      Root CA certificate (PEM format) used to verify the container registry certificate over HTTPS (if the container registry uses self-signed SSL certificates).

    - **spec.registry.dockerCfg**
      string

      Container registry access token in Base64 (`~/.docker/config.json` format). Leave this field empty if anonymous access to the container registry is used.

    - **spec.registry.login**
      string

      Username for authenticating to the container registry.

    - **spec.registry.password**
      string

      Password for authenticating to the container registry.

    - **spec.registry.repo**
      string

      Required value

      Address of the package repository in the container registry.

    - **spec.registry.scheme**

string

Protocol for accessing the repository (for example, `https`).

- **spec.scanInterval**
  string

  Interval for container registry scan.

  Defines the frequency of checking the container registry for new packages.

  Default: `6h`

  Pattern: `^(\d+h)?(\d+m)?(\d+s)?$`

  Examples:

  ```
  scanInterval: 5m
  ```

  ```
  scanInterval: 1h
  ```

  ```
  scanInterval: 6h30m
  ```

- **status**
  object

  Package repository status.

  - **status.conditions**
    array of objects

    Conditions reflecting the latest observations of the repository state.

    Condition describing one aspect of the current repository state.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

    - **status.conditions.message**
      string

      Required value

      message is a human readable message indicating details about the transition. This may be an empty string.

      Maximum length: `32768`

    - **status.conditions.observedGeneration**
      integer

      observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

      Allowed values: `0 <= X`

    - **status.conditions.reason**
      string

      Required value

      reason contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

      Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

      Length: `1..1024`

    - **status.conditions.status**
      string

      Required value

      status of the condition, one of True, False, Unknown.

      Allowed values: `True`, `False`, `Unknown`

    - **status.conditions.type**
      string

Required value

type of condition in CamelCase or in foo.example.com/CamelCase.

Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

Maximum length: `316`

- **status.lastChangeTime**
  string

  Time of the most recent scan that found at least one new version. Scans that found nothing new do not advance this timestamp.

- **status.lastNewVersions**
  integer

  Number of new versions found by the most recent scan. Set to zero when the last scan found nothing new.

- **status.lastScanTime**
  string

  Time of the most recent scan of any outcome.

- **status.message**
  string

  Human-readable message about the repository status.

- **status.packages**
  array of objects

  List of packages available in this repository.

  - **status.packages.name**
    string

    Required value

    Name of the package.

  - **status.packages.type**
    string

    Required value

    Type of the package.

- **status.packagesCount**
  integer

  Total number of packages in this repository.

- **status.partialScanAvailable**
  boolean

  Indicates whether the container registry supports pagination when listing tags.

- **status.phase**
  string

  Current phase of the repository.

PackageRepositoryOperation

Short names: pro

Scope: Cluster
Version: v1alpha1

PackageRepositoryOperation represents an operation to scan/update a package repository.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info in the Kubernetes documentation

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info in the Kubernetes documentation

- **metadata**
  object
- **spec**
  object

Required value

Defines parameters for the package repository operation.

- **spec.packageRepositoryName**
  string

  Required value

  Name of the package repository to operate on.

- **spec.type**
  string

  Required value

  Type of operation to perform.

  Allowed values: `Update`

- **spec.update**
  object

  Configuration for update operations.

  - **spec.update.fullScan**
    boolean

    Whether to perform a full scan of the repository.

  - **spec.update.timeout**
    string

    Timeout for the operation.

- **status**
  object

  Package repository operation status.

  - **status.completionTime**
    string

    Time when the operation completed.

  - **status.conditions**
    array of objects

    Conditions reflecting the latest observations of the operation state. The operation phase is determined by the `Completed` condition: while its status is `False`, the operation is in one of the intermediate phases (`Discover`, `Processing`); when the status is `True`, the operation has finished with reason `Succeeded` or `Failed`.

    Condition describing one aspect of the current operation state.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

    - **status.conditions.message**
      string

      Required value

      message is a human readable message indicating details about the transition. This may be an empty string.

      Maximum length: `32768`

    - **status.conditions.observedGeneration**
      integer

      observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

      Allowed values: `0 <= X`

    - **status.conditions.reason**
      string

      Required value

      Programmatic identifier for the reason of the condition's last transition. For the `Completed` condition, the values are: `Discover`, `Processing`, `Succeeded`, `Failed`.

      Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  status of the condition, one of True, False, Unknown.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  Condition type. This resource uses the `Completed` condition, which reflects whether the operation has reached a terminal phase.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.packages**
  object

  Information about packages processed during the operation.

  - **status.packages.discovered**
    array of objects

    List of packages discovered during the operation.

    - **status.packages.discovered.name**
      string

      Required value

      Name of the discovered package.

  - **status.packages.failed**
    array of objects

    List of packages that failed processing.

    - **status.packages.failed.errors**
      array of objects

      Required value

      List of errors encountered while processing this package.

      - **status.packages.failed.errors.message**
        string

        Required value

        Message of the error.

      - **status.packages.failed.errors.version**
        string

        Required value

        Version of the package that failed.

    - **status.packages.failed.name**
      string

      Required value

      Name of the package that failed.

  - **status.packages.newVersionsOverall**
    integer

    Total number of newly found versions across all packages in this operation.

    A version is counted as new if its ApplicationPackageVersion or ModulePackageVersion did not exist in the cluster, or (ApplicationPackageVersion only) existed with the “not in registry” mark and its image was found in the registry during this operation.

  - **status.packages.processed**
    array of objects

    List of packages successfully processed.

    - **status.packages.processed.foundVersions**
      integer

      Number of versions found during this operation.

        - **status.packages.processed.name**
          string

          Required value

          Name of the processed package.

        - **status.packages.processed.newVersions**
          integer

          Number of newly found versions during this operation.

          A version is counted as new if its ApplicationPackageVersion or ModulePackageVersion did not exist in the cluster, or (ApplicationPackageVersion only) existed with the “not in registry” mark and its image was found in the registry during this operation.

        - **status.packages.processed.type**
          string

          Type of the package.

      - **status.packages.processedOverall**
        integer

        Total number of packages processed.

      - **status.packages.total**
        integer

        Total number of packages found.

  - **status.startTime**
    string

    Time when the operation started.

SSHConfig

Version: dhctl.deckhouse.io/v1

General dhctl SSH config.

Example:

```
apiVersion: dhctl.deckhouse.io/v1
kind: SSHConfig
sshUser: user
sshPort: 22
sshExtraArgs: "-vvv"
sshAgentPrivateKeys:
- key: "<ssh-private-key>"
```

- **apiVersion**
  string

  Version of the Deckhouse API.

  Allowed values: `dhctl.deckhouse.io/v1`

- **kind**
  string

  Allowed values: `SSHConfig`

- **legacyMode**
  boolean

  Switch to legacy SSH mode (clissh).

- **modernMode**
  boolean

  Switch to modern SSH mode (gossh).

- **sshAgentPrivateKeys**
  array of objects
  - **sshAgentPrivateKeys.key**
    string

    Required value

    Private SSH key.

  - **sshAgentPrivateKeys.passphrase**
    string

    Password for SSH key.

- **sshBastionHost**
  string

  SSH bastion host.

- **sshBastionPassword**
  string

  A password for the bastion user.

- **sshBastionPort**
  integer

  Port of SSH bastion.

- **sshBastionUser**
  string

  Username for bastion.

- **sshExtraArgs**
  string

  Additional arguments for SSH connection.

- **sshPort**
  integer

  SSH port.

- **sshUser**
  string

  SSH username.

- **sudoPassword**
  string

  A sudo password for the user.

SSHHost

Version: dhctl.deckhouse.io/v1

General dhctl SSH host config.

Example:

```
apiVersion: dhctl.deckhouse.io/v1
kind: SSHHost
host: 172.16.0.0
```

- **apiVersion**
  string

  Required value

  Version of the Deckhouse API.

  Allowed values: `dhctl.deckhouse.io/v1`

- **host**
  string

  Required value

  Host.

- **kind**
  string

  Required value

  Allowed values: `SSHHost`

StaticClusterConfiguration

Version: deckhouse.io/v1

Parameters of a static (bare metal) cluster.

To change the `StaticClusterConfiguration` resource in a running cluster, run the following command:

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

Example:

```
apiVersion: deckhouse.io/v1
kind: StaticClusterConfiguration
internalNetworkCIDRs:
- 10.244.0.0/16
- 10.50.0.0/16
```

- **apiVersion**
  string

  Required value

  Version of the Deckhouse API.

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **internalNetworkCIDRs**
  array of strings

  List of internal cluster networks.

  Internal cluster networks connect Kubernetes components (`kube-apiserver`, `kubelet`, etc.).

  The parameter is mandatory in the following cases:

  - Cluster nodes have more than one network interface
  - The cluster is deployed inside a Deckhouse Virtualization Platform

  > **i** When changing the value, subnets in the new list must include IP addresses of currently joined nodes.

  Example:

  ```
  internalNetworkCIDRs:
  - 192.168.42.0/24
  - 172.16.16.0/24
  ```

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **kind**
  string

  Required value

  Allowed values: `StaticClusterConfiguration`

Release channels

> **i** For information on which versions of Deckhouse are available on which release channels as well as the planned date of the version update for a particular release channel, visit releases.deckhouse.io website.

Clusters, as infrastructure elements, usually have to meet various requirements.

A production cluster, unlike a development one, has higher requirements for reliability. In a production cluster, frequent component updates and changes are undesirable. All the cluster components must be thoroughly tested for stable and reliable operation.

Deckhouse uses **five release channels** which you can *soft-switch* between using the deckhouse module: just specify the desired release channel in the module configuration.

- Rock Solid
- Stable
- Early Access
- Beta
- Alpha

- The least stable release channel but with the most frequent appearance of new versions. Can bring new features but can be unstable.
- For more broad testing of new features to catch regressions
- **Recommended** release channel. Is mostly safe and can be used in non-critical environments or for local development
- Is mostly safe and we encourage you to use this version everywhere
- The most stable release channel and recommended for usage in critical environments with tight SLA

Additional resources

- releases.deckhouse.io
- Module `deckhouse` documentation

Supported Kubernetes and OS versions

Linux

The following Linux distributions are currently supported for nodes:

Linux Distribution	Supported versions	Supported in redactions	Notes
		CE, BE, SE, SE+, EE	
CentOS	**7, 8, 9**	✓	
Debian	**10, 11, 12, 13**	✓	
openSUSE	**15.4, 15.5, 15.6**	✓	
Rocky Linux	**8, 9**	✓	
Ubuntu	**18.04, 20.04, 22.04, 24.04**	✓	

Kubernetes

The following Kubernetes versions are currently supported:

Version	Status
1.31	The version is not recommended for use. It is planned for removal in future releases.
1.32	Production-ready fully supported version.
1.33	**Default version.** Production-ready fully supported version.
1.34	Production-ready fully supported version.
1.35	Early access version, which is not suitable for production environments.

Container registry

> **i** Deckhouse Kubernetes Platform works with any container registry supporting Basic or Bearer authentication and OCI image format specification.

Container registries that Deckhouse is guaranteed to work with
Sonatype Nexus Repository
Harbor
JFrog Artifactory
CNCF Distribution Registry (Docker Registry)
Project Quay*

Deckhouse Kubernetes Platform editions comparison

The Deckhouse Kubernetes Platform is available in several editions. They are with different functions and support options. You can find a brief comparison of editions listing its main features and functions on the Deckhouse Kubernetes Platform website.

The table below provides listing of the Deckhouse Kubernetes Platform modules available in each edition:

	CE	BE	SE	SE+	EE
adaptive-resource-management	✓	✓	✓	✓	✓

	CE	BE	SE	SE+	EE
admission-policy-engine	~ Restriction on the ability to use signature verification policies for container images and the ability to prohibit the launch of containers with vulnerabilities	~ Restriction on the ability to use signature verification policies for container images and the ability to prohibit the launch of containers with vulnerabilities	~ Restriction on the ability to use signature verification policies for container images and the ability to prohibit the launch of containers with vulnerabilities	~ Restriction on the ability to use signature verification policies for container images and the ability to prohibit the launch of containers with vulnerabilities	✓
ai-models	✗	✗	✗	✗	✓
alb	✓	✓	✓	✓	✓
ansible	✓	✗	✗	✓	✓
cert-manager	✓	✓	✓	✓	✓
chrony	✓	✓	✓	✓	✓
cilium-hubble	✓	✓	✓	✓	✓
cloud-provider-aws	✓	✓	✓	✓	✓
cloud-provider-azure	✓	✓	✓	✓	✓
cloud-provider-dvp	✓	✓	✓	✓	✓
cloud-provider-dynamix	✗	✗	✗	✗	✓
cloud-provider-gcp	✓	✓	✓	✓	✓
cloud-provider-huaweicloud	✗	✗	✗	✗	✓
cloud-provider-openstack	✗	✗	✗	✗	✓
cloud-provider-vcd	✗	✗	✗	✗	✓
cloud-provider-vsphere	✗	✗	✗	✓	✓
cloud-provider-yandex	✓	✓	✓	✓	✓
cloud-provider-zvirt	✗	✗	✗	✓	✓
cni-cilium	~ Restriction on the ability to configure fault tolerant Egress Gateways	~ Restriction on the ability to configure fault tolerant Egress Gateways	~ Restriction on the ability to configure fault tolerant Egress Gateways	✓	✓
cni-flannel	✓	✓	✓	✓	✓
cni-simple-bridge	✓	✓	✓	✓	✓
code	✗	✗	~ A separate license is required	~ A separate license is required	~ A separate license is required

	CE	BE	SE	SE+	EE
commander	~ A separate license is required	~ A separate license is required	~ A separate license is required	✓	✓
commander-agent	✓	✓	✓	✓	✓
console	✓	✓	✓	✓	✓
control-plane-manager	✓	✓	✓	✓	✓
csi-ceph	✓	✗	✓	✓	✓
csi-hpe	✗	✗	✗	✗	✓
csi-huawei	✗	✗	✗	✗	✓
csi-netapp	✗	✗	✗	✗	✓
csi-nfs	✓	✗	✓	✓	✓
csi-s3	✗	✗	✗	✗	✓
csi-scsi-generic	✗	✗	✗	✗	✓
csi-vsphere	✗	✗	✗	✓	✓
csi-yadro-tatlin-unified	✗	✗	✗	✗	✓
dashboard	✓	✓	✗	✗	✓
deckhouse	✓	✓	✓	✓	✓
deckhouse-tools	✓	✓	✓	✓	✓
descheduler	✓	✓	✓	✓	✓
development-platform	✗	✗	✗	✗	✓
documentation	✓	✓	✓	✓	✓
extended-monitoring	✓	✓	✓	✓	✓
gpu	✗	✗	✗	✗	✓
ingress-nginx	✓	✓	✓	✓	✓
istio	~ Restriction on the use of federation and multicluster features	~ Restriction on the use of federation and multicluster features	~ Restriction on the use of federation and multicluster features	~ Restriction on the use of federation and multicluster features	✓
keepalived	✗	✗	✗	✗	✓
kube-dns	✓	✓	✓	✓	✓
kube-proxy	✓	✓	✓	✓	✓
local-path-provisioner	✓	✓	✓	✓	✓
log-shipper	✓	✓	✓	✓	✓
loki	✓	✓	✓	✓	✓
managed-cassandra	✗	✗	✗	✗	✓
managed-clickhouse	✗	✗	✗	✗	✓
managed-hive-metastore	✗	✗	✗	✗	✓
managed-kafka	✗	✗	✗	✗	✓
managed-memcached	✗	✗	✓	✓	✓
managed-opensearch	✗	✗	✗	✗	✓
managed-postgres	✗	✗	✓	✓	✓
managed-starrocks	✗	✗	✓	✓	✓
managed-trino	✗	✗	✗	✗	✓
managed-valkey	✗	✗	✓	✓	✓
metallb	✗	✗	~ Restriction on the use of the BGP load balancer for services	~ Restriction on the use of the BGP load balancer for services	✓
monitoring-custom	✓	✓	✓	✓	✓
monitoring-deckhouse	✓	✓	✓	✓	✓
monitoring-kubernetes	✓	✓	✓	✓	✓
monitoring-kubernetes-control-plane	✓	✓	✓	✓	✓
monitoring-ping	✓	✓	✓	✓	✓

	CE	BE	SE	SE+	EE
multitenancy-manager	✓	✓	✓	✓	✓
namespace-configurator	✓	✓	✓	✓	✓
network-gateway	✗	✗	✗	✗	✓
network-policy-engine	✓	✓	✓	✓	✓
neuvector	✗	✗	✗	✗	✓
node-local-dns	✗	✓	✓	✓	✓
node-manager	✓	✓	✓	✓	✓
observability	✓	✓	✓	✓	✓
observability-platform	✗	✗	✗	✗	~ A separate license is required
okmeter	✓	✓	✓	✓	✓
openvpn	✓	✓	✓	✓	✓
operator-argo	✗	✗	✗	✗	✓
operator-helm	✓	✓	✓	✓	✓
operator-prometheus	✓	✓	✓	✓	✓
operator-trivy	✗	✗	✗	✗	✓
payload-registry	✗	✗	✗	✗	✓
pod-reloader	✓	✓	✓	✓	✓
prometheus	✓	✓	✓	✓	✓
prometheus-metrics-adapter	✓	✓	✓	✓	✓
prometheus-pushgateway	✓	✓	✓	✓	✓
prompp	✓	✓	✓	✓	✓
registry	✓	✓	✓	✓	✓
registry-packages-proxy	✓	✓	✓	✓	✓
runtime-audit-engine	✗	✗	✗	✗	✓
sdn	✗	✗	✗	✗	✓
sds-elastic	✗	✗	✗	✗	✓
sds-local-volume	~ Restriction on the ability to create snapshots and volume wiping	✗	✓	✓	✓
sds-node-configurator	✓	✗	✓	✓	✓
sds-object	✗	✗	✗	✗	✓
sds-replicated-volume	~ Restriction on the ability to create snapshots	✗	✓	✓	✓
secret-copier	✓	✓	✓	✓	✓
secrets-store-integration	✓	✓	✓	✓	✓
security-events-manager	✗	✗	✗	✗	✓
service-with-healthchecks	✗	✗	✗	✗	✓
snapshot-controller	✓	✓	✓	✓	✓
state-snapshotter	✓	✓	✓	✓	✓
static-routing-manager	✗	✗	✗	✗	✓
storage-foundation	✓	✓	✓	✓	✓
storage-volume-data-manager	✗	✗	✗	✗	✓

	CE	BE	SE	SE+	EE
stronghold	~ Some features are available depending on the selected Deckhouse Stronghold license. For more details, see the "Editions" section of the Deckhouse Stronghold documentation.	~ Some features are available depending on the selected Deckhouse Stronghold license. For more details, see the "Editions" section of the Deckhouse Stronghold documentation.	~ Some features are available depending on the selected Deckhouse Stronghold license. For more details, see the "Editions" section of the Deckhouse Stronghold documentation.	~ Some features are available depending on the selected Deckhouse Stronghold license. For more details, see the "Editions" section of the Deckhouse Stronghold documentation.	~ Some features are available depending on the selected Deckhouse Stronghold license. For more details, see the "Editions" section of the Deckhouse Stronghold documentation.
terraform-manager	✓	✓	✓	✓	✓
upmeter	✓	✓	✓	✓	✓
user-authn	✓	✓	✓	✓	✓
user-authz	~ Restriction on the use of namespace filtering in the role-based access model	✓	✓	✓	✓
vertical-pod-autoscaler	✓	✓	✓	✓	✓
virtualization	~ There are limitations to functionality	✗	✗	✓	✓

Container registry structure

This page provides a general overview of the container image registries used to install the Deckhouse Kubernetes Platform (DKP).

Commercial editions container registry

The registry contains modules and components available in the commercial DKP editions:

- EE (address `registry.deckhouse.io/deckhouse/ee`).
- SE (address `registry.deckhouse.io/deckhouse/se`).
- SE+ (address `registry.deckhouse.io/deckhouse/se-plus`).
- BE (address `registry.deckhouse.io/deckhouse/be`).

Key features:

- implements functionality according to the edition;
- security testing is performed as part of the secure software development lifecycle;
- delivered electronically.

Images are provided with technical support, including operational consultation.

Image build and update specifics:

- built from open-source software included in DKP:
  - from upstream (as-is);
  - with deep modernization (fork).
- includes proprietary company developments;
- vulnerability remediation and package updates are performed only when upstream open-source packages are updated;
- updates are provided:
  - with new functionality;
  - with bug and vulnerability fixes;
  - with security incident resolution.

Network interaction of the platform components

If the infrastructure where Deckhouse Kubernetes Platform (DKP) is running has requirements to limit host-to-host network communications, the following conditions must be met:

- Tunneling mode for traffic between pods is enabled (configuration for CNI Cilium, configuration for CNI Flannel).
- Traffic between podSubnetCIDR encapsulated within a VXLAN is allowed (if inspection and filtering of traffic within a VXLAN tunnel is performed).
- If there is integration with external systems (e.g. LDAP, SMTP or other external APIs), it is required to allow network communication with them.
- Local network communication is fully allowed within each individual cluster node.
- Inter-node communication is allowed on the ports shown in the tables on the current page. Note that most ports are in the 4200-4299 range. When new platform components are added, they will be assigned ports from this range (if it is possible).

How to check the current VXLAN port...

```
d8 k -n d8-cni-cilium get cm cilium-config -o yaml | grep tunnel
```

Example output:

```
routing-mode: tunnel
tunnel-port: "4298"
tunnel-protocol: vxlan
```

> **i** Changes related to the addition, removal, or reassignment of ports in the tables are listed in the “Network” section of a respective DKP version on the Release notes page.

Requirements for network latency between master nodes

> **i** DKP supports HA mode using arbiter nodes. The network latency requirements listed below also apply to arbiter nodes.

Etcd only commits a writing after the change has been replicated to a majority of master nodes. Therefore, the round-trip time (RTT) between master nodes directly determines the cluster’s performance. A single operation in the cluster typically consists of several consecutive writes, so as latency increases, the application of manifests, operator operations, and Kubernetes API responsiveness slow down noticeably.

The sum of RTT and jitter between master nodes must not exceed 100 ms.

> ⚠ Round-trip time (RTT) is specified, not one-way delay: for example, the `ping` command returns RTT.
>
> Jitter—the variation in latency—is added to the RTT. For example, with an RTT of 90 ms and jitter of 20 ms, the total is 110 ms, and the requirement is not met, even though the RTT value itself is within the 100 ms limit.

When planning a cluster, it is recommended to aim for a latency of no more than 50 ms between future master nodes—that is, a twofold margin relative to the requirement. This margin is necessary for the following reasons:

- Pre-deployment measurements are performed on an idle network and may underestimate actual values.
- You need to account for latency spikes and variation, as well as possible infrastructure operations—for example, migrating a virtual machine to a zone with higher latency.

Where does the 100 ms value come from...

DKP uses the standard etcd values:

- `ETCD_HEARTBEAT_INTERVAL` — 100 ms, the frequency at which the leader sends heartbeat messages to the other cluster members.
- `ETCD_ELECTION_TIMEOUT` — 1000 ms, the timeout for heartbeat messages; after this time, a cluster member begins leader re-election.

The etcd documentation requires that the election timeout be at least 10x RTT. Hence, the maximum allowed value is:

```
1000 ms / 10 = 100 ms
```

Potential issues

The main consequence of increased RTT is a proportional slowdown of the cluster. Network latency is directly reflected in write times: with an RTT of 300 ms, each write to etcd will take at least 300 ms. There is no threshold here; performance degrades continuously as latency increases.

The second issue is etcd leader re-elections. These are triggered not by the magnitude of the latency, but by its complete loss: a cluster member initiates an election if it does not receive messages from the leader within the time specified by `ETCD_ELECTION_TIMEOUT` (taking into account etcd's built-in randomization mechanism—from 1,000 to 2,000 ms). With a 100-ms heartbeat interval, this corresponds to 10–20 consecutive missed heartbeat messages. Typical causes:

- A network outage or packet loss lasting 1–2 seconds.
- Disk write delays on the leader node, preventing etcd from sending heartbeat messages in time.
- A restart of the etcd process.

Isolated spikes in latency do not trigger re-elections: heartbeat messages arrive, albeit late. Only writes that occur during the spike are delayed.

However, a high RTT makes re-elections protracted when they do occur. Votes are collected within a single RTT and must fit within `ETCD_ELECTION_TIMEOUT`. If the RTT is close to this timeout, the election does not complete in time and restarts, leaving the cluster leaderless. A tenfold margin is required to ensure the election succeeds on the first attempt.

During leader re-elections, etcd does not process write requests, so the Kubernetes API returns errors. The `KubeEtcdHighNumberOfLeaderChanges` alert signals frequent leader re-elections—more than three leader re-elections in 10 minutes.

Checking current latencies

Latency and its variation can be measured using `ping`. The command is run on one of the master nodes, or—when planning the cluster—on the server intended to become a master node. ICMP is sufficient for this check, as it is already included in the requirements for traffic between nodes:

```
ping -C 100 -i 0.2 <IP address of another master node>
```

Example of the resulting output:

```
rtt min/avg/max/mdev = 25.085/25.502/35.933/1.185 ms
```

Here, `avg` is the average RTT, `mdev` is the delay variation (jitter), and `max` is the actual maximum—that is, the RTT including spikes. There is no need to add the RTT and jitter manually: you should compare the `max` value against the requirements. When planning a cluster, we recommend staying within 50 ms; for a live cluster, do not exceed 100 ms.

Network latency at the other nodes

The control plane has no requirements regarding network latency on nodes that are not master nodes: latency on these nodes does not affect the cluster's stability. The only factor that matters is the duration of the loss of communication with master nodes—by default, after 40 seconds, a node enters the `Unreachable` state, and after 5 minutes, its pods are migrated to other nodes. Both thresholds are configured using the nodeMonitorGracePeriodSeconds and failedNodePodEvictionTimeoutSeconds parameters.

These delays affect the performance of everything running on the nodes—including the platform's own components. For components that access the Kubernetes API (DKP operators, Prometheus, Ingress controllers), latency to the master nodes is added to every request. Latency between nodes is added to every network request between pods and to DNS lookups, and—when using replicated storage—to the time it takes to write to disk.

Thus, the acceptable value is determined by the requirements of the running workload. For clusters distributed across remote sites, this means that the platform will continue to operate, but application latencies will reflect the geographic distribution of the nodes.

Traffic between master nodes

Port	Protocol	Purpose

Port	Protocol	Purpose
`2379, 2380`	TCP	etcd replication
`4200`	TCP	Cluster API webhook handler
`4201`	TCP	Webhook handler for VMware Cloud Director cloud provider
`4223`	TCP	Deckhouse controller webhook handler

Traffic from master nodes to regular nodes

Port	Protocol	Purpose
`22`	TCP	SSH for static node bootstrapping by the static provider
`10250`	TCP	kubelet
`4221`	TCP	`bashible apiserver` for delivering node configurations
`4227`	TCP	`runtime-audit-engine` webhook handler

Traffic from nodes to master nodes

Port	Protocol	Purpose
`4234`	UDP	NTP for time synchronization between nodes
`6443`	TCP	`kube-apiserver` for controllers working in the node’s host network namespace
`4203`	TCP	`machine-controller-manager` metrics
`4219`	TCP	Proxy for registry packages `registry-packages-proxy`
`4222`	TCP	Deckhouse controller metrics

Traffic between nodes

Port	Protocol	Purpose
—	ICMP	ICMP for node-to-node connectivity monitoring
`4202`	TCP	`sds-node-configurator` node agent metrics
`4204`	TCP	Deckhouse controller debug

Port	Protocol	Purpose
4205	TCP	ebpf-exporter module metrics
4206	TCP	node-exporter module metrics
4207	TCP	ingress-nginx controller metrics for the HostWithFailover inlet
4208	TCP	ingress-nginx controller metrics for the HostWithFailover inlet
4209	TCP	Kubernetes control plane metrics
4210	TCP	kube-proxy metrics
4211	TCP	Cluster API metrics
4212	TCP	runtime-audit-engine module metrics
4213	TCP	kube-router metrics
4214	TCP	API of the sds-replicated-volume module agent
4215	TCP	sds-replicated-volume agent metrics
4216	TCP	storage-status agent metrics
4218	TCP/UDP	Synchronization of the speaker components in metallb modules via memberlist protocol
4220	TCP	Metrics of the speaker components in metallb modules
4224	TCP	node-local-dns metrics
4225	TCP/UDP	Synchronization of the speaker components in metallb modules via memberlist protocol
4226	TCP	Metrics of the speaker components in metallb modules
4228	TCP	sds-node-configurator agent healthcheck
4229	TCP	csi-nfs CSI controller healthcheck
4230	TCP	csi-nfs CSI node healthcheck
4231	TCP	csi-hpe CSI controller healthcheck

Port	Protocol	Purpose
4232	TCP	csi-hpe CSI node healthcheck
4235	TCP	csi-s3 CSI controller healthcheck
4236	TCP	csi-s3 CSI node healthcheck
4237	TCP	csi-scsi-generic CSI controller healthcheck
4238	TCP	csi-scsi-generic CSI node healthcheck
4239	TCP	storage-status agent healthcheck
4240	TCP	CNI Cilium node-to-node healthcheck
4241	TCP	CNI Cilium agent metrics
4242	TCP	CNI Cilium operator metrics
4244	TCP	cilium-hubble module API
4245	TCP	chrony-exporter metrics
4246	TCP	csi-ceph CephFS CSI controller healthcheck
4247	TCP	csi-ceph RBD CSI controller healthcheck
4248	TCP	csi-yadro-tatlin-unified CSI controller healthcheck
4249	TCP	csi-yadro-tatlin-unified CSI node healthcheck
4250	TCP	sds-local-volume CSI controller healthcheck
4251	TCP	sds-local-volume CSI node healthcheck
4252	TCP	csi-ceph RBD CSI node healthcheck
4253	TCP	csi-ceph CephFS CSI node healthcheck
4254	TCP	csi-netapp CSI controller healthcheck
4255	TCP	csi-netapp CSI node healthcheck
4256	TCP	csi-netapp CSI controller metrics

Port	Protocol	Purpose
`4257`	TCP	`csi-netapp` CSI controller API port
`4258`	TCP	`csi-huawei` CSI controller webhook port
`4259`	TCP	`csi-huawei` CSI node healthcheck
`4260`	TCP	`csi-huawei` CSI controller metrics
`4261`	TCP	`sds-replicated-volume` CSI controller healthcheck
`4262`	TCP	`sds-replicated-volume` CSI node healthcheck
`4263`	TCP	`service-with-healthchecks` module metrics
`4269`	TCP	`sds-replicated-volume` CSI node healthcheck
`4270`	TCP	`sds-replicated-volume` CSI node metrics
`4271`	TCP	`sds-replicated-volume` CSI controller healthcheck
`4272`	TCP	`sds-replicated-volume` CSI controller metrics
`4135-4199`	TCP	`virtualization` live migration tunnels between cluster nodes
`4280`	TCP	`virtualization` USB over IP (usb/ip) protocol for sharing USB devices between cluster nodes
`4286`	TCP	Istio CNI metrics
`4287`	UDP	WireGuard port for traffic encryption in CNI Cilium
`4288`	TCP	`monitoring-ping` metrics
`4289`	TCP	`monitoring-ping` metrics
`4295-4297`	UDP	Used by the `cni-cilium` module for VXLAN encapsulation of inter-pod traffic in multiple nested virtualization — when DKP with the `virtualization` module enabled is deployed inside virtual machines that are also created in DKP with the `virtualization` module enabled

Port	Protocol	Purpose
`4298`	UDP	Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods if the cluster was deployed on DKP version starting from **1.71** (for clusters deployed on DKP versions up to **1.71**, see the note for ports `4299/UDP`, `8469/UDP`, and `8472/UDP`)
`4299`	UDP	**For clusters deployed on DKP versions 1.64–1.70.** Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods. Updating DKP to newer versions will not change the port used unless the `virtualization` module is enabled. **Note** that in such clusters, enabling the `virtualization` module on DKP up to version 1.70 changes the port to `4298/UDP`
`7000-7999`	TCP	`sds-replicated-volume` DRBD replication
`8469`	UDP	**For clusters deployed on DKP version 1.63 and below with the `virtualization` module enabled prior to DKP version 1.63.** Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods. Updating DKP to newer versions will not change the occupied port
`8472`	UDP	**For clusters deployed on DKP version 1.63 and below.** Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods. Updating DKP to newer versions will not change the occupied port if the `virtualization` module is not enabled. **Note** that in such clusters, enabling the `virtualization` module on DKP before version 1.70 changes the port: • Enabling the `virtualization` module on DKP version 1.63 and below will change it to `8469/UDP` and will not change with subsequent DKP updates • Enabling the `virtualization` module on DKP starting from version 1.64 will change it to `4298/UDP` and will not change with subsequent DKP updates

Static `virtualization` ports on cluster nodes (`4100-4134/TCP` reserved)

Port	Protocol	Purpose
`4100`	TCP	`virtualization` `virt-handler` healthz and metrics
`4101`	TCP	`virtualization` `virt-handler` console server
`4105`	TCP	`virtualization` `vm-route-forge` liveness and readiness probes

Port	Protocol	Purpose
`4106`	TCP	`virtualization` `vm-route-forge` pprof in debug mode
`4107`	TCP	`virtualization` `virtualization-dra` gRPC liveness and readiness probes

External traffic to master nodes

Port	Protocol	Purpose
`22`	TCP	SSH for initializing Deckhouse Kubernetes Platform
`6443`	TCP	Direct access to the apiserver

External traffic to frontend nodes

Port	Protocol	Purpose
`80, 443`	TCP	Application ports for requests to Ingress controllers over HTTP and HTTPS. Note that these ports are configurable in the IngressNginxController resource and may vary across installations
`5416`	UDP	OpenVPN
`5416`	TCP	OpenVPN
`10256`	TCP	Healthcheck port for external balancers
`30000-32767`	TCP	NodePort range

External traffic for all nodes

Port	Protocol	Purpose
`53`	UDP	DNS
`53`	TCP	DNS
`123`	UDP	NTP for synchronization with external time servers
`179`	TCP	BGP peering with the infrastructure router for network traffic balancing (metallb module)
`443`	TCP	Container registry

Additional resources

- Global custom resources
- Module `cni-cilium` parameters
- Module `cni-flannel` parameters
- Module `control-plane-manager` parameters

The list of alerts

The page displays a list of all alerts of monitoring in the Deckhouse Kubernetes Platform.

> **i** The list does not contain alerts of modules from a source.

Alerts are grouped by modules. To the right of the alert name, there are icons indicating the minimum DKP edition in which the alert is available, and the alert severity.

For each alert, a summary is provided, and if available, the detailed alert description can be viewed by expanding it.

Alert severity

Alert descriptions contain the Severity (S) parameter, which indicates the level of criticality. Its value varies from `S1` to `S9` and can be interpreted as follows:

- `S1` — maximum level, critical failure/crash (immediate action required);
- `S2` — high level, close to maximum, possible accident (rapid response required);
- `S3` — medium level, potentially serious problem (verification required);
- `S4`-`S9` — low level. There is a problem, but overall performance is not impaired.

Module admission-policy-engine

- D8AdmissionPolicyEngineNotBootstrapped CE S7
  ▼ Admission-policy-engine module hasn't been bootstrapped for 10 minutes.

  The admission-policy-engine module couldn’t bootstrap.

  Steps to troubleshoot:

  1. Verify that the module’s components are up and running:

     ```
     d8 k get pods -n d8-admission-policy-engine
     ```

  2. Check logs for issues, such as missing constraint templates or incomplete CRD creation:

     ```
     d8 k logs -n d8-system -lapp=deckhouse --tail=1000 | grep admission-policy-engine
     ```

- OperationPolicyViolation CE S3
  ▼ At least one object violates the configured cluster operation policies.

  You have configured operation policies for the cluster, and one or more existing objects are violating these policies.

  To identify violating objects:

  - Run the following Prometheus query:

    ```
    count by (violating_namespace, violating_kind, violating_name,
    violation_msg) (
      d8_gatekeeper_exporter_constraint_violations{
        violation_enforcement="deny",
        source_type="OperationPolicy"
      }
    )
    ```

  - Alternatively, check the admission-policy-engine Grafana dashboard.

- PodSecurityStandardsViolation CE S3
  ▼ At least one pod violates the configured cluster pod security standards.

  You have configured Pod Security Standards, and one or more running pods are violating these standards.

  To identify violating pods:

  - Run the following Prometheus query:

```
count by (violating_namespace, violating_name, violation_msg) (
  d8_gatekeeper_exporter_constraint_violations{
    violation_enforcement="deny",
    violating_namespace=~".*",
    violating_kind="Pod",
    source_type="PSS"
  }
)
```

  - Alternatively, check the admission-policy-engine Grafana dashboard.

- SecurityPolicyViolation CE S3
  ▼ At least one object violates the configured cluster security policies.

  You have configured security policies for the cluster, and one or more existing objects are violating these policies.

  To identify violating objects:

  - Run the following Prometheus query:

```
count by (violating_namespace, violating_kind, violating_name,
violation_msg) (
  d8_gatekeeper_exporter_constraint_violations{
    violation_enforcement="deny",
    source_type="SecurityPolicy"
  }
)
```

  - Alternatively, check the admission-policy-engine Grafana dashboard.

Module cert-manager

- CertmanagerCertificateChallengePending CE S4
  ▼ ACME Challenge for domain "XXX" is pending for more than 1 hour.

  The ACME Challenge `NAMESPACE/NAME` for domain `XXX` has been in `pending` state for more than 1 hour and requires investigation.

  Common causes:

  - DNS misconfiguration (for DNS-01 challenges).
  - Ingress or firewall issues blocking HTTP-01 validation.
  - ACME provider rate limiting.
  - Network connectivity issues.

  Resolution steps:

  1. Check the Challenge status and reason:

```
d8 k -n NAMESPACE describe challenge NAME
```

  2. Check `cert-manager` logs for errors:

```
d8 k -n d8-cert-manager logs -l app=cert-manager -c cert-manager --tail=100
```

  3. If the challenge is stuck, delete the Order to trigger a retry:

```
d8 k -n NAMESPACE get orders
d8 k -n NAMESPACE delete order ORDER_NAME
```

- CertmanagerCertificateChallengePending CE S6
  ▼ ACME Challenge for domain "XXX" is pending for more than 10 minutes.

  The ACME Challenge `NAMESPACE/NAME` for domain `XXX` has been in `pending` state for more than 10 minutes.

  This may indicate issues with the ACME challenge validation process.

Common causes:

- DNS propagation delays (for DNS-01 challenges).
- Ingress or firewall issues blocking HTTP-01 validation.
- ACME provider rate limiting or temporary unavailability.

Diagnostic steps:

1. Check the Challenge status:

   ```
   d8 k -n NAMESPACE describe challenge NAME
   ```

2. Check `cert-manager` logs for errors:

   ```
   d8 k -n d8-cert-manager logs -l app=cert-manager -c cert-manager --tail=100
   ```

- CertmanagerCertificateChallengeStuck CE S4
  ▾ ACME Challenge for domain "XXX" is stuck in "STATUS_REFERENCE" state.

  The ACME Challenge `NAMESPACE/NAME` for domain `XXX` is in `STATUS_REFERENCE` state and requires manual intervention.

  Reason: XXX

  The certificate will not be issued or renewed until the issue is resolved.

  Common causes:

  - DNS misconfiguration (for DNS-01 challenges).
  - Ingress or firewall issues (for HTTP-01 challenges).
  - Rate limiting by the ACME provider.
  - Authorization expired before challenge completion.

  Resolution steps:

  1. Check the Challenge status and reason:

     ```
     d8 k -n NAMESPACE describe challenge NAME
     ```

  2. Delete the failed Order to trigger a new certificate request:

     ```
     d8 k -n NAMESPACE get orders
     d8 k -n NAMESPACE delete order ORDER_NAME
     ```

- CertmanagerCertificateExpired CE S4
  ▾ Certificate NAMESPACE/NAME is not provided.

  Certificate is not provided.

  To check the certificate details, run the following command:

  ```
  d8 k -n NAMESPACE describe certificate NAME
  ```

- CertmanagerCertificateExpiredSoon CE S4
  ▾ Certificate will expire soon.

  The certificate `NAMESPACE/NAME` will expire in less than two weeks.

  To check the certificate details, run the following command:

  ```
  d8 k -n NAMESPACE describe certificate CERTIFICATE-NAME
  ```

- CertmanagerCertificateOrderErrors CE S5
  ▾ Cert-manager couldn't order a certificate.

  Cert-manager received responses with the status code `STATUS_REFERENCE` when requesting `SCHEME://HOST_NAMEPATH`.

  This can affect certificate ordering and prolongation in the future. For details, check the cert-manager logs using the following command:

  ```
  d8 k -n d8-cert-manager logs -l app=cert-manager -c cert-manager
  ```

Module chrony

- NodeTimeOutOfSync CE S5
  ▼ Clock on the node NODE_NAME is drifting.

  Time on the node `NODE_NAME` is out of sync and drifts apart from the NTP server clock by VALUE seconds.

  To resolve the time synchronization issues:

  - Fix network errors:
    - Ensure the upstream time synchronization servers defined in the chrony configuration are available.
    - Eliminate large packet loss and excessive latency to upstream time synchronization servers.
  - Modify the NTP servers list defined in the chrony configuration.
- NTPDaemonOnNodeDoesNotSynchronizeTime CE S5
  ▼ NTP daemon on the node NODE_NAME haven't synchronized time for too long.

  Steps to troubleshoot:

  1. Check if the chrony pod is running on the node by executing the following command:

     ```
     d8 k -n d8-chrony --field-selector spec.nodeName="NODE_NAME" get pods
     ```

  2. Verify the chrony daemon’s status by executing the following command:

     ```
     d8 k -n d8-chrony exec POD_NAME -- /opt/chrony-static/bin/chronyc sources
     ```

  3. Resolve the time synchronization issues:

     - Fix network errors:
       - Ensure the upstream time synchronization servers defined in the chrony configuration are available.
       - Eliminate large packet loss and excessive latency to upstream time synchronization servers.
     - Modify the NTP servers list defined in the chrony configuration.

Module cloud-provider-huaweicloud

- D8HuaweiCloudMachineStuckInDeleting EE S6
  ▼ HuaweiCloudMachine NAME is stuck in Deleting for more than 24h.

  The HuaweiCloudMachine `NAME` in namespace `NAMESPACE` has been stuck in Deleting state for more than 24 hours.

  This usually means the deletion job in HuaweiCloud is failing or the API is unavailable.

  To inspect the machine status, run:

  ```
  d8 k get huaweicloudmachine -n NAMESPACE NAME -o yaml
  ```

  Check the controller logs for errors:

  ```
  d8 k logs -n d8-cloud-provider-huaweicloud -l app=caphc-controller-manager
  ```

Module cloud-provider-yandex

- D8YandexNatInstanceConnectionsQuotaUtilization CE S4
  ▼ Connection quota utilization of the Yandex NAT instance exceeds 85% over the last 5 minutes.

  The connection quota for the Yandex NAT instance has exceeded 85% utilization over the past 5 minutes.

  To prevent potential issues, contact Yandex technical support and request an increase in the connection quota.

- NATInstanceWithDeprecatedAvailabilityZone CE S9
  ▼ NAT instance NAME is in a deprecated availability zone.

The NAT instance `NAME` is located in the availability zone `ru-central1-c`, which has been deprecated by Yandex Cloud. To resolve this issue, migrate the NAT instance to either `ru-central1-a` or `ru-central1-b` by following the instructions below.

> **i** The migration process involves irreversible changes and may result in a significant downtime. The duration typically depends on Yandex Cloud's response time and can last several tens of minutes.

1. Migrate the NAT instance. To get `providerClusterConfiguration.withNATInstance`, run the following command:

   ```
   d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values -g cloud-provider-yandex -o json | jq -c | jq '.cloudProviderYandex.internal.providerClusterConfiguration.withNATInstance'
   ```

   - If you specified `withNATInstance.natInstanceInternalAddress` and/or `withNATInstance.internalSubnetID` in `providerClusterConfiguration`, remove them using the following command:

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

   - If you specified `withNATInstance.externalSubnetID` and/or `withNATInstance.natInstanceExternalAddress` in `providerClusterConfiguration`, change them to the appropriate values.

     To get the address and subnet ID, use the Yandex Cloud Console or CLI.

     To change `withNATInstance.externalSubnetID` and `withNATInstance.natInstanceExternalAddress`, run the following command:

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

2. Run the appropriate edition and version of the Deckhouse installer container on the **local** machine. You may have to change the container registry address to do that. After that, perform the converge.

   - Get the appropriate edition and version of Deckhouse:

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

   - Run the installer:

     ```
     docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
     ```

   - Perform the converge:

     ```
     dhctl converge --ssh-agent-private-keys=/tmp/.ssh/SSH_KEY_FILENAME --ssh-user=USERNAME --ssh-host MASTER-NODE-0-HOST
     ```

3. Update the route table.

   - Get the route table name:

     ```
     d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values -g cloud-provider-yandex -o json | jq -c | jq '.global.clusterConfiguration.cloud.prefix'
     ```

- Get the NAT instance name:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values -g cloud-provider-yandex -o json | jq -c | jq '.cloudProviderYandex.internal.providerDiscoveryData.natInstanceName'
```

- Get the NAT instance internal IP address:

```
yc compute instance list | grep -e "INTERNAL IP" -e NAT_INSTANCE_NAME_FROM_PREVIOUS_STEP
```

- Update the route:

```
yc vpc route-table update --name ROUTE_TABLE_NAME_FROM_PREVIOUS_STEP --route "destination=0.0.0.0/0,next-hop=NAT_INSTANCE_INTERNAL_IP_FROM_PREVIOUS_STEP"
```

- NodeGroupNodeWithDeprecatedAvailabilityZone CE S9
  ▼ **NodeGroup NODE_GROUP_NAME contains nodes in a deprecated availability zone.**

  Certain nodes in the node group `NODE_GROUP_NAME` are located in the availability zone `ru-central1-c`, which has been deprecated by Yandex Cloud.

  Steps to troubleshoot:

  1. Identify the nodes that need to be migrated by running the following command:

     ```
     d8 k get node -l "topology.kubernetes.io/zone=ru-central1-c"
     ```

  2. Migrate your nodes, disks, and load balancers to one of the supported zones: `ru-central1-a`, `ru-central1-b`, or `ru-central1-d`. Refer to the Yandex migration guide for detailed instructions.

     > **i** You can’t migrate public IP addresses between zones. For details, refer to the migration guide.

Module cni-cilium

- CiliumAgentEndpointsNotReady CE S4
  ▼ **Over 50% of all known endpoints aren't ready in agent NAMESPACE/POD_NAME.**

  For details, refer to the logs of the agent:

  ```
  d8 k -n NAMESPACE logs POD_NAME
  ```

- CiliumAgentMapPressureCritical CE S4
  ▼ **eBPF map MAP_NAME exceeds 90% utilization in agent NAMESPACE/POD_NAME.**

  The eBPF map resource utilization limit has almost been reached.

  Check with the vendor for potential remediation steps.

- CiliumAgentMetricNotFound CE S4
  ▼ **Agent NAMESPACE/POD_NAME isn't sending some metrics.**

  Steps to troubleshoot:

  1. Check the logs of the agent:

     ```
     d8 k -n NAMESPACE logs POD_NAME
     ```

  2. Verify the agent’s health status:

     ```
     d8 k -n NAMESPACE exec -ti POD_NAME cilium-health status
     ```

3. Compare the metrics with those of a neighboring agent.

> **i** Note that the absence of metrics can indirectly indicate that new pods can’t be created on the node due to connectivity issues with the agent.

- CiliumAgentPolicyImportErrors CE S4
  ▼ Agent NAMESPACE/POD_NAME fails to import policies.

  For details, refer to the logs of the agent:

  ```
  d8 k -n NAMESPACE logs POD_NAME
  ```

- CiliumAgentUnreachableHealthEndpoints CE S4
  ▼ Agent NAMESPACE/POD_NAME can't reach some of the node health endpoints.

  For details, query Cilium health status API:

  ```
  d8 k -n NAMESPACE exec POD_NAME -- cilium-health status
  ```

  You can also refer to the agent logs:

  ```
  d8 k -n NAMESPACE logs POD_NAME
  ```

- CniCiliumNonStandardVXLANPortFound CE S4
  ▼ Cilium configuration uses a non-standard VXLAN port.

  The Cilium configuration specifies a non-standard VXLAN port `XXX`. The recommended port number is `XXX`.

  To resolve this issue, update the `tunnel-port` parameter in the `cilium-configmap` ConfigMap located in the `d8-cni-cilium` namespace to match the recommended port.

  If you configured the non-standard port on purpose, ignore this alert.

- CniCiliumOrphanEgressGatewayPolicyFound SE-PLUS S4
  ▼ Orphaned EgressGatewayPolicy with an irrelevant EgressGateway name has been found.

  The cluster contains an orphaned EgressGatewayPolicy named `NAME` with an irrelevant EgressGateway name.

  To resolve this issue, verify the EgressGateway name specified in the EgressGatewayPolicy resource `EGRESSGATEWAY` and update it as needed.

- D8NodeHasUnmetKernelRequirements CE S4
  ▼ Nodes have unmet kernel requirements.

  Deckhouse has detected that some nodes don’t meet the required kernel constraints. As a result, certain modules can’t run on these nodes.

  Kernel requirements for each module:

  - **Cilium module**: Kernel version must be **>= 5.8**.

  To list all affected nodes, use the `d8_node_kernel_does_not_satisfy_requirements == 1` expression in Prometheus.

Module control-plane-manager

- D8ControlPlaneManagerPodNotRunning CE S6
  ▼ Controller pod isn't running on node NODE_NAME.

  The `d8-control-plane-manager` pod is either failing or hasn’t been scheduled on node `NODE_NAME`.

  To resolve this issue, check the status of the `kube-system/d8-control-plane-manager` DaemonSet and its pods by running the following command:

  ```
  d8 k -n kube-system get daemonset,pod --selector=app=d8-control-plane-manager
  ```

- D8EtcdDatabaseHighFragmentationRatio CE S7

▼ etcd database size in use is less than 50% of the allocated storage.

The etcd database size in use on instance `INSTANCE_NAME` is less than 50% of the allocated disk space, indicating potential fragmentation. Additionally, the total storage size exceeds 75% of the configured quota.

To resolve this issue, defragment the etcd database by running the following command:

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt \
  --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

- D8EtcdExcessiveDatabaseGrowth CE S4
  ▼ etcd cluster database is growing rapidly.

  Based on the growth rate observed over the last six hours, Deckhouse predicts that the etcd database will run out of disk space within one day on instance `INSTANCE_NAME`.

  To prevent disruptions, investigate the cause and take necessary action.

- D8EtcdHighRequestsLatency CE S5
  ▼ Etcd request latency (p99) has exceeded 800 ms multiple times in the past hour.

  The 99th percentile of etcd request duration on `INSTANCE_NAME` has exceeded 800 ms more than twice in the last hour.

  High etcd latency can cause API server timeouts, slow admission webhooks, and general cluster instability.

  Investigate the following:

  - **Node disk I/O**: Etcd is extremely sensitive to disk performance. Check disk utilization and I/O wait on the control plane node.
  - **Network-attached storage**: If the etcd data directory resides on NFS, Ceph, or another network storage, check connectivity and latency to the storage backend. Etcd requires a consistently low-latency local disk (SSD recommended).
  - **Control plane summary**: Review the **Kubernetes Cluster → Control Plane Status** Grafana dashboard for an overall picture of API server and etcd health, request rates, and error rates.
  - **Node base performance**: Check node CPU and RAM utilization and load average. If the node is CPU-bound (load average significantly exceeds the number of CPU cores), consider scaling up the control plane node.
- D8KubeEtcdDatabaseSizeCloseToTheLimit CE S3
  ▼ etcd database size is approaching the limit.

  The etcd database size on `NODE_NAME` is nearing its size limit. This may be caused by a high number of events, such as pod evictions or the recent creation of numerous resources in the cluster.

  Possible solutions:

  - Defragment the etcd database by running the following command:

    ```
    d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
      --cacert /etc/kubernetes/pki/etcd/ca.crt \
      --cert /etc/kubernetes/pki/etcd/ca.crt \
      --key /etc/kubernetes/pki/etcd/ca.key \
      --endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
    ```

  - Increase node memory. Starting from 24 GB, `quota-backend-bytes` will increase by 1 GB for every extra 8 GB of memory.

    Example:

Node memory	quota-backend-bytes
16 GB	2147483648 (2 GB)
24 GB	3221225472 (3 GB)

Node memory	quota-backend-bytes
32 GB	4294967296 (4 GB)
40 GB	5368709120 (5 GB)
48 GB	6442450944 (6 GB)
56 GB	7516192768 (7 GB)
64 GB	8589934592 (8 GB)
72 GB	8589934592 (8 GB)
…	…

- D8KubernetesStaleTokensDetected CE S8
  ▾ **Stale service account tokens detected for XXX/XXX.**

  Service account `XXX/XXX` is using a stale token. This issue may occur if an application reads the token only at startup and does not reload it periodically. As a result, an outdated token might be used, leading to security breach and authentication failures.

  Recommended actions:

  - Ensure your application is configured to periodically reload the token from the file system.
  - Verify that you are using an up-to-date client library that supports automatic token rotation.

  Note that currently these tokens are not blocked because the `--service-account-extend-token-expiration` flag is enabled by default (`Default: true`). With this flag enabled, admission-injected tokens are extended up to 1 year during token generation to facilitate a safe transition from legacy tokens to the bound service account token feature, ignoring the value of `service-account-max-token-expiration`.

  For further investigation: You can find the problematic pods using the following command:

  ```
  d8 k get pods -A -o json | jq -r '.items[] | select(.spec.serviceAccountName=="XXX"
  and .metadata.namespace=="XXX") | .metadata.name'
  ```

- D8KubernetesVersionIsDeprecated CE S7
  ▾ **Kubernetes version VERSION_NUMBER is deprecated.**

  The current Kubernetes version `VERSION_NUMBER` has been deprecated, and support for it will be removed in upcoming releases.

  Please migrate to the next kubernetes version (at least 1.32)

  Read documentation about how to update the Kubernetes version in the cluster.

- D8NeedDecreaseEtcdQuotaBackendBytes CE S6
  ▾ **Deckhouse suggests reducing quota-backend-bytes.**

  When the control plane node memory is reduced, Deckhouse may suggest reducing `quota-backend-bytes`. While Deckhouse is capable of automatically increasing this value, reducing it must be done manually.

  To modify `quota-backend-bytes`, set the `controlPlaneManager.etcd.maxDbSize` parameter. Before setting a new value, check the current database usage on every control plane node by running:

  ```
  for pod in $(d8 k get pod -n kube-system -l component=etcd,tier=control-plane -o name); do
    d8 k -n kube-system exec -ti "$pod" -- /usr/bin/etcdctl \
      --cacert /etc/kubernetes/pki/etcd/ca.crt \
      --cert /etc/kubernetes/pki/etcd/ca.crt \
      --key /etc/kubernetes/pki/etcd/ca.key \
      endpoint status -w json | jq --arg a "$pod" -r \
      '.[0].Status.dbSize / 1024 / 1024 | tostring | $a + ": " + . + " MB"';
  done
  ```

  Things to note:

  - The maximum value for `controlPlaneManager.etcd.maxDbSize` is 8 GB.
  - If control plane nodes have less than 24 GB, set `controlPlaneManager.etcd.maxDbSize` to 2 GB.
  - Starting from 24 GB, `quota-backend-bytes` will increase by 1 GB for every extra 8 GB of memory.

Example:

Node memory	quota-backend-bytes
16 GB	2147483648 (2 GB)
24 GB	3221225472 (3 GB)
32 GB	4294967296 (4 GB)
40 GB	5368709120 (5 GB)
48 GB	6442450944 (6 GB)
56 GB	7516192768 (7 GB)
64 GB	8589934592 (8 GB)
72 GB	8589934592 (8 GB)
…	…

- D8ProblematicFeatureGateInUse CE S9
  ▼ Feature gate XXX is STATUS_REFERENCE in current Kubernetes version XXX and should be removed from the configuration. Is not using in the cluster control-plane.

  The feature gate `XXX` is currently enabled in the control-plane-manager ModuleConfig, but it is STATUS_REFERENCE in the current Kubernetes version `XXX`.

  XXX This feature gate does not exist in any component (apiserver, kube-controller-manager, kube-scheduler, kubelet) and is not being applied to the cluster. XXX

  Recommended actions:

  1. Remove `XXX` from the `spec.settings.enabledFeatureGates` list in the control-plane-manager ModuleConfig:

  ```
  d8 k edit mc control-plane-manager
  ```

  For more information about Kubernetes feature gates, refer to the official documentation.

- D8UsingFeatureGateWillBeDeprecated CE S9
  ▼ Feature gate XXX will be deprecated in Kubernetes version XXX.

  The feature gate `XXX` is currently enabled in the control-plane-manager ModuleConfig, but it will be deprecated in Kubernetes version `XXX`.

  Current Kubernetes version: `XXX`

  Recommended actions:

  1. Review the usage of this feature gate in your cluster.
  2. Plan migration away from this feature gate before upgrading to version `XXX`.
  3. Remove `XXX` from the `spec.settings.enabledFeatureGates` list in the control-plane-manager ModuleConfig:

  ```
  d8 k edit mc control-plane-manager
  ```

  For more information about Kubernetes feature gates, refer to the official documentation.

Module documentation

- ModuleConfigDeprecated CE S9
  ▼ Deprecated deckhouse-web ModuleConfig detected.

  The `deckhouse-web` module has been renamed to `documentation`, and a new `documentation` ModuleConfig is generated automatically.

  Steps to troubleshoot:

  1. Remove the deprecated `deckhouse-web` ModuleConfig from the CI deployment process.
  2. Delete it using the following command:

     ```
     d8 k delete mc deckhouse-web
     ```

- ModuleDocumentationRenderError CE S6

▾ Documentation rendering failed for module MODULE_NAME.

An error occurred while rendering documentation for the `MODULE_NAME` module. The broken module was removed from the documentation site during Hugo build.

Check the `ModuleDocumentation` resource status for details:

```
d8 k get moduledocumentations MODULE_NAME -o yaml
```

Check docs-builder logs for more information:

```
d8 k -n d8-system logs -l app=docs-builder | grep -i "removed broken module"
```

Module extended-monitoring

- CertificateSecretExpired CE S8
  ▾ Certificate has expired.

  A certificate in Secret `SECRET_NAMESPACE/SECRET_NAME` has expired.

  Ways to resolve:

  - If the certificate is managed manually, upload a new certificate.
  - If the certificate is managed by the `cert-manager` module, inspect the certificate resource:

    1. Retrieve the certificate name from the Secret:

       ```
       cert=$(d8 k get secret -n SECRET_NAMESPACE SECRET_NAME -o 'jsonpath=
       {.metadata.annotations.cert-manager\.io/certificate-name}')
       ```

    2. Check the certificate status and investigate why it hasn’t been updated:

       ```
       d8 k describe cert -n SECRET_NAMESPACE "$cert"
       ```

- CertificateSecretExpiredSoon CE S8
  ▾ Certificate is expiring soon.

  A certificate in Secret `SECRET_NAMESPACE/SECRET_NAME` will expire in less than two weeks.

  Ways to resolve:

  - If the certificate is managed manually, upload a new certificate.
  - If the certificate is managed by the `cert-manager` module, inspect the certificate resource:

    1. Retrieve the certificate name from the Secret:

       ```
       cert=$(d8 k get secret -n SECRET_NAMESPACE SECRET_NAME -o 'jsonpath=
       {.metadata.annotations.cert-manager\.io/certificate-name}')
       ```

    2. Check the certificate status and investigate why it hasn’t been updated:

       ```
       d8 k describe cert -n SECRET_NAMESPACE "$cert"
       ```

- CronJobAuthenticationFailure CE S7
  ▾ Unable to log in to the container registry using imagePullSecrets for the IMAGE_NAME image.

  Deckhouse was unable to log in to the container registry using `imagePullSecrets` for the `IMAGE_NAME` image.

  To resolve this issue, investigate the possible causes in the following sources:

  - The `NAMESPACE` namespace.
  - The CronJob `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- CronJobAuthorizationFailure CE S7
  ▾ Insufficient privileges to pull the IMAGE_NAME image using the specified imagePullSecrets.

  Deckhouse has insufficient privileges to pull the `IMAGE_NAME` image using the specified

`imagePullSecrets`.

To resolve this issue, investigate the possible causes in the following sources:

- The `NAMESPACE` namespace.
- The CronJob `NAME`.
- The `CONTAINER_NAME` container in the registry.

- CronJobBadImageFormat CE S7
  ▼ **The IMAGE_NAME image name is incorrect.**

  Deckhouse has detected that the `IMAGE_NAME` image name is incorrect.

  To resolve this issue, check that the `IMAGE_NAME` image name is spelled correctly in the following sources:

  - The `NAMESPACE` namespace.
  - The CronJob `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- CronJobFailed CE S5
  ▼ **Job NAMESPACE/JOB_NAME failed in CronJob NAMESPACE/CRONJOB.**

  Deckhouse has detected that Job `NAMESPACE/JOB_NAME` failed in CronJob `NAMESPACE/CRONJOB`.

  Steps to resolve:

  1. Print the job details:

     ```
     d8 k -n NAMESPACE describe job JOB_NAME
     ```

  2. Check the job status:

     ```
     d8 k -n NAMESPACE get job JOB_NAME
     ```

  3. Check the status of pods created by the job:

     ```
     d8 k -n NAMESPACE get pods -l job-name=JOB_NAME
     ```

- CronJobImageAbsent CE S7
  ▼ **The IMAGE_NAME image is missing from the registry.**

  Deckhouse has detected that the `IMAGE_NAME` image is missing from the container registry.

  To resolve this issue, check whether the `IMAGE_NAME` image is available in the following sources:

  - The `NAMESPACE` namespace.
  - The CronJob `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- CronJobImageAvailabilityUnknownError CE S7
  ▼ **An unknown error occurred with the IMAGE_NAME image availability check.**

  Deckhouse has detected an unknown error while checking `IMAGE_NAME` image availability in the following sources:

  - The `NAMESPACE` namespace.
  - The CronJob `NAME`.
  - The `CONTAINER_NAME` container in the registry.

  Review the image-availability-exporter logs to see the detailed error information:

  ```
  d8 k -n d8-monitoring logs -l app=image-availability-exporter -c image-availability-exporter
  ```

- CronJobPodsNotCreated CE S5
  ▼ **Pods set in CronJob NAMESPACE/JOB_NAME haven't been created.**

  Deckhouse has detected that the pods set in CronJob `NAMESPACE/CRONJOB` still haven’t been created.

  Steps to resolve:

  1. Print the job details:

     ```
     d8 k -n NAMESPACE describe job JOB_NAME
     ```

2. Check the job status:

   ```
   d8 k -n NAMESPACE get job JOB_NAME
   ```

3. Check the status of pods created by the job:

   ```
   d8 k -n NAMESPACE get pods -l job-name=JOB_NAME
   ```

- CronJobRegistryUnavailable CE S7
  ▼ **The container registry is not available for the IMAGE_NAME image.**

  Deckhouse has detected that the container registry is not available for the `IMAGE_NAME` image.

  To resolve this issue, investigate the possible causes in the following sources:

  - The `NAMESPACE` namespace.
  - The CronJob `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- CronJobSchedulingError CE S6
  ▼ **CronJob NAMESPACE/CRONJOB failed to schedule on time.**

  Deckhouse has detected that CronJob `NAMESPACE/CRONJOB` failed to schedule on time.

  - Current schedule: `XXX`
  - Last scheduled time: `XXX%`
  - Next projected schedule time: `XXX%`
- D8CertExporterPodIsNotReady CE S8
  ▼ **The x509-certificate-exporter pod isn't ready.**

  Steps to resolve:

  1. Retrieve the deployment details:

     ```
     d8 k -n d8-monitoring describe deploy x509-certificate-exporter
     ```

  2. Check the pod status and investigate why it’s not ready:

     ```
     d8 k -n d8-monitoring describe pod -l app=x509-certificate-exporter
     ```

- D8CertExporterPodIsNotRunning CE S8
  ▼ **The x509-certificate-exporter pod isn't running.**

  Steps to resolve:

  1. Retrieve the deployment details:

     ```
     d8 k -n d8-monitoring describe deploy x509-certificate-exporter
     ```

  2. Check the pod status and investigate why it’s not running:

     ```
     d8 k -n d8-monitoring describe pod -l app=x509-certificate-exporter
     ```

- D8CertExporterTargetAbsent CE S8
  ▼ **There is no x509-certificate-exporter target in Prometheus.**

  Ways to resolve:

  - Check the pod status:

    ```
    d8 k -n d8-monitoring get pod -l app=x509-certificate-exporter
    ```

  - Check the pod logs:

    ```
    d8 k -n d8-monitoring logs -l app=x509-certificate-exporter -c x509-certificate-exporter
    ```

- D8CertExporterTargetDown CE S8
  ▼ **Prometheus can't scrape x509-certificate-exporter metrics.**

  Ways to resolve:

  - Check the pod status:

    ```
    d8 k -n d8-monitoring get pod -l app=x509-certificate-exporter
    ```

  - Check the pod logs:

    ```
    d8 k -n d8-monitoring logs -l app=x509-certificate-exporter -c x509-certificate-exporter
    ```

- D8ImageAvailabilityExporterMalfunctioning CE S8
  ▼ **The image-availability-exporter has crashed.**

  The `image-availability-exporter` has failed to perform any image availability checks in the container registry for over 20 minutes.

  To investigate the issue, review the exporter's logs:

  ```
  d8 k -n d8-monitoring logs -l app=image-availability-exporter -c image-availability-exporter
  ```

- D8ImageAvailabilityExporterPodIsNotReady CE S8
  ▼ **The image-availability-exporter pod is not ready.**

  Deckhouse has detected that the `image-availability-exporter` pod is not ready. As a result, the images listed in the `image` field aren't checked for availability in the container registry.

  Steps to resolve:

  1. Retrieve the deployment details:

     ```
     d8 k -n d8-monitoring describe deploy image-availability-exporter
     ```

  2. Check the pod status and investigate why it isn't `Ready`:

     ```
     d8 k -n d8-monitoring describe pod -l app=image-availability-exporter
     ```

- D8ImageAvailabilityExporterPodIsNotRunning CE S8
  ▼ **The image-availability-exporter pod is not running.**

  Deckhouse has detected that the `image-availability-exporter` pod is not running. As a result, the images listed in the `image` field aren't checked for availability in the container registry.

  Steps to resolve:

  1. Retrieve the deployment details:

     ```
     d8 k -n d8-monitoring describe deploy image-availability-exporter
     ```

  2. Check the pod status and investigate why it isn't running:

     ```
     d8 k -n d8-monitoring describe pod -l app=image-availability-exporter
     ```

- D8ImageAvailabilityExporterTargetAbsent CE S8
  ▼ **The image-availability-exporter target is missing from Prometheus.**

  Deckhouse has detected that the `image-availability-exporter` target is missing from Prometheus.

  Steps to resolve:

  1. Check the pod status:

     ```
     d8 k -n d8-monitoring get pod -l app=image-availability-exporter
     ```

  2. Check the pod logs:

     ```
     d8 k -n d8-monitoring logs -l app=image-availability-exporter -c image-availability-
     exporter
     ```

- D8ImageAvailabilityExporterTargetDown CE S8

▾ Prometheus can't scrape metrics of image-availability-exporter.

Deckhouse has detected that Prometheus is unable to scrape metrics of `image-availability-exporter`.

Steps to resolve:

1. Check the pod status:

   ```
   d8 k -n d8-monitoring get pod -l app=image-availability-exporter
   ```

2. Check the pod logs:

   ```
   d8 k -n d8-monitoring logs -l app=image-availability-exporter -c image-availability-exporter
   ```

- DaemonSetAuthenticationFailure CE S7
  ▾ Unable to log in to the container registry using imagePullSecrets for the IMAGE_NAME image.

  Deckhouse was unable to log in to the container registry using `imagePullSecrets` for the `IMAGE_NAME` image.

  To resolve this issue, investigate the possible causes in the following sources:

  - The `NAMESPACE` namespace.
  - The DaemonSet `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- DaemonSetAuthorizationFailure CE S7
  ▾ Insufficient privileges to pull the IMAGE_NAME image using the specified imagePullSecrets.

  Deckhouse has insufficient privileges to pull the `IMAGE_NAME` image using the specified `imagePullSecrets`.

  To resolve this issue, investigate the possible causes in the following sources:

  - The `NAMESPACE` namespace.
  - The DaemonSet `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- DaemonSetBadImageFormat CE S7
  ▾ The IMAGE_NAME image name is incorrect.

  Deckhouse has detected that the `IMAGE_NAME` image name is incorrect.

  To resolve this issue, check that the `IMAGE_NAME` image name is spelled correctly in the following sources:

  - The `NAMESPACE` namespace.
  - The DaemonSet `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- DaemonSetImageAbsent CE S7
  ▾ The IMAGE_NAME image is missing from the registry.

  Deckhouse has detected that the `IMAGE_NAME` image is missing from the container registry.

  To resolve this issue, check whether the `IMAGE_NAME` image is available in the following sources:

  - The `NAMESPACE` namespace.
  - The DaemonSet `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- DaemonSetImageAvailabilityUnknownError CE S7
  ▾ An unknown error occurred with the IMAGE_NAME image availability check.

  Deckhouse has detected an unknown error while checking `IMAGE_NAME` image availability in the following sources:

  - The `NAMESPACE` namespace.
  - The DaemonSet `NAME`.
  - The `CONTAINER_NAME` container in the registry.

  Review the image-availability-exporter logs to see the detailed error information:

  ```
  d8 k -n d8-monitoring logs -l app=image-availability-exporter -c image-availability-exporter
  ```

- DaemonSetRegistryUnavailable CE S7
  ▾ **The container registry is not available for the IMAGE_NAME image.**

  Deckhouse has detected that the container registry is not available for the `IMAGE_NAME` image.

  To resolve this issue, investigate the possible causes in the following sources:

  - The `NAMESPACE` namespace.
  - The DaemonSet `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- DeploymentAuthenticationFailure CE S7
  ▾ **Unable to log in to the container registry using imagePullSecrets for the IMAGE_NAME image.**

  Deckhouse was unable to log in to the container registry using `imagePullSecrets` for the `IMAGE_NAME` image.

  To resolve this issue, investigate the possible causes in the following sources:

  - The `NAMESPACE` namespace.
  - The Deployment `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- DeploymentAuthorizationFailure CE S7
  ▾ **Insufficient privileges to pull the IMAGE_NAME image using the specified imagePullSecrets.**

  Deckhouse has insufficient privileges to pull the `IMAGE_NAME` image using the specified `imagePullSecrets`.

  To resolve this issue, investigate the possible causes in the following sources:

  - The `NAMESPACE` namespace.
  - The Deployment `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- DeploymentBadImageFormat CE S7
  ▾ **The IMAGE_NAME image name is incorrect.**

  Deckhouse has detected that the `IMAGE_NAME` image name is incorrect.

  To resolve this issue, check that the `IMAGE_NAME` image name is spelled correctly in the following sources:

  - The `NAMESPACE` namespace.
  - The Deployment `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- DeploymentImageAbsent CE S7
  ▾ **The IMAGE_NAME image is missing from the registry.**

  Deckhouse has detected that the `IMAGE_NAME` image is missing from the container registry.

  To resolve this issue, check whether the `IMAGE_NAME` image is available in the following sources:

  - The `NAMESPACE` namespace.
  - The Deployment `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- DeploymentImageAvailabilityUnknownError CE S7
  ▾ **An unknown error occurred with the IMAGE_NAME image availability check.**

  Deckhouse has detected an unknown error while checking `IMAGE_NAME` image availability in the following sources:

  - The `NAMESPACE` namespace.
  - The Deployment `NAME`.
  - The `CONTAINER_NAME` container in the registry.

  Review the image-availability-exporter logs to see the detailed error information:

  ```
  d8 k -n d8-monitoring logs -l app=image-availability-exporter -c image-availability-exporter
  ```
- DeploymentRegistryUnavailable CE S7
  ▾ **The container registry is not available for the IMAGE_NAME image.**

  Deckhouse has detected that the container registry is not available for the `IMAGE_NAME` image.

  To resolve this issue, investigate the possible causes in the following sources:

  - The `NAMESPACE` namespace.
  - The Deployment `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- ExtendedMonitoringTargetDown CE S5
  ▾ Extended monitoring is unavailable.

  The pod running `extended-monitoring-exporter` is currently unavailable.

  As a result, the following alerts will not be triggered:

  - Low disk space and inode usage on volumes.
  - CPU overloads and container throttling.
  - `500` errors on Ingress.
  - Insufficient replicas of Deployments, StatefulSets, and DaemonSets.
  - Other alerts associated with this exporter.

  To resolve this issue, investigate its possible causes:

  1. Print detailed information about the `extended-monitoring-exporter` deployment:

     ```
     d8 k -n d8-monitoring describe deploy extended-monitoring-exporter
     ```

  2. Print detailed information about the pods associated with the `extended-monitoring-exporter`:

     ```
     d8 k -n d8-monitoring describe pod -l app=extended-monitoring-exporter
     ```

- IngressResponses5xx CE S4
  ▾ URL VHOST/LOCATION on Ingress INGRESS has more than XXX% of 5xx responses from the backend.

  Deckhouse has detected that URL VHOST/LOCATION on Ingress `INGRESS`, using service `SERVICE_NAME` on port SERVICE_PORT has more than XXX% of `5xx` responses from the backend.

  Current rate of `5xx` responses: VALUE%

- IngressResponses5xx CE S5
  ▾ URL VHOST/LOCATION on Ingress INGRESS has more than XXX% of 5xx responses from the backend.

  Deckhouse has detected that URL VHOST/LOCATION on Ingress `INGRESS`, using service `SERVICE_NAME` on port SERVICE_PORT, has more than XXX% of `5xx` responses from the backend.

  Current rate of `5xx` responses: VALUE%

- KubernetesDaemonSetNotUpToDate CE S9
  ▾ There were VALUE outdated pods in DaemonSet NAMESPACE/DAEMONSET_NAME over the last 15 minutes.

  Deckhouse has detected VALUE outdated pods in DaemonSet `NAMESPACE/DAEMONSET_NAME` over the last 15 minutes.

  Steps to resolve:

  1. Check the DaemonSet’s status:

     ```
     d8 k -n NAMESPACE get ds DAEMONSET_NAME
     ```

  2. Analyze the DaemonSet’s description:

     ```
     d8 k -n NAMESPACE describe ds DAEMONSET_NAME
     ```

  3. If the parameter `Number of Nodes Scheduled with Up-to-date Pods` does not match `Current Number of Nodes Scheduled`, check the DaemonSet’s `updateStrategy`:

     ```
     d8 k -n NAMESPACE get ds DAEMONSET_NAME -o json | jq '.spec.updateStrategy'
     ```

     If `updateStrategy` is set to `OnDelete`, the DaemonSet is updated only when pods are deleted.

- KubernetesDaemonSetReplicasUnavailable CE S5

▾ No available replicas remaining in DaemonSet NAMESPACE/DAEMONSET_NAME.

Deckhouse has detected that there are no available replicas remaining in DaemonSet `NAMESPACE/DAEMONSET_NAME`.

List of unavailable pods:

```
XXXXXX, XXXPOD_NAMEXXX
```

If you know where the DaemonSet should be scheduled, run the command below to identify the problematic nodes. Use a label selector for pods, if needed.

```
d8 k -n NAMESPACE get pod -ojson | jq -r '.items[] |
select(.metadata.ownerReferences[] | select(.name =="DAEMONSET_NAME")) |
select(.status.phase != "Running" or ([ .status.conditions[] | select(.type ==
"Ready" and .status == "False") ] | length ) == 1 ) |
.spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.node
SelectorTerms[].matchFields[].values[]'
```

- KubernetesDaemonSetReplicasUnavailable CE S6
  ▾ The number of unavailable replicas in DaemonSet NAMESPACE/DAEMONSET_NAME exceeds the threshold.

  Deckhouse has detected that the number of unavailable replicas in DaemonSet `NAMESPACE/DAEMONSET_NAME` exceeds the threshold.

  - Current number: `VALUE` unavailable replica(s).
  - Threshold number: `XXX` unavailable replica(s).

  List of unavailable pods:

  ```
  XXXXXX, XXXPOD_NAMEXXX
  ```

  If you know where the DaemonSet should be scheduled, run the command below to identify the problematic nodes. Use a label selector for pods, if needed.

  ```
  d8 k -n NAMESPACE get pod -ojson | jq -r '.items[] |
  select(.metadata.ownerReferences[] | select(.name =="DAEMONSET_NAME")) |
  select(.status.phase != "Running" or ([ .status.conditions[] | select(.type ==
  "Ready" and .status == "False") ] | length ) == 1 ) |
  .spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.node
  SelectorTerms[].matchFields[].values[]'
  ```

- KubernetesDeploymentReplicasUnavailable CE S5
  ▾ No available replicas remaining in deployment NAMESPACE/DEPLOYMENT_NAME.

  Deckhouse has detected that there are no available replicas remaining in deployment `NAMESPACE/DEPLOYMENT_NAME`.

  List of unavailable pods:

  ```
  XXXXXX, XXXPOD_NAMEXXX
  ```

- KubernetesDeploymentReplicasUnavailable CE S6
  ▾ The number of unavailable replicas in deployment NAMESPACE/DEPLOYMENT_NAME exceeds spec.strategy.rollingupdate.maxunavailable.

  Deckhouse has detected that the number of unavailable replicas in deployment `NAMESPACE/DEPLOYMENT NAME` exceeds the value set in `spec.strategy.rollingupdate.maxunavailable`.

  - Current number: `VALUE` unavailable replica(s).
  - Threshold number: `XXX` unavailable replica(s).

  List of unavailable pods:

  ```
  XXXXXX, XXXPOD_NAMEXXX
  ```

- KubernetesStatefulSetReplicasUnavailable CE S5
  ▾ No ready replicas remaining in StatefulSet NAMESPACE/STATEFULSET.

Deckhouse has detected that there are no ready replicas remaining in StatefulSet `NAMESPACE/STATEFULSET`.

List of unavailable pods:

```
XXXXXX, XXXPOD_NAMEXXX
```

- KubernetesStatefulSetReplicasUnavailable CE S6
  ▼ The number of unavailable replicas in StatefulSet NAMESPACE/STATEFULSET exceeds the threshold.

  Deckhouse has detected that the number of unavailable replicas in StatefulSet `NAMESPACE/STATEFULSET` exceeds the threshold.

  - Current number: `VALUE` unavailable replica(s).
  - Threshold number: `XXX` unavailable replica(s).

  List of unavailable pods:

  ```
  XXXXXX, XXXPOD_NAMEXXX
  ```

- LoadAverageHigh CE S4
  ▼ Average load on node NODE_NAME is too high.

  Over the last 5 minutes, the average load on node `NODE_NAME` has been higher than XXX per core.

  There are more processes in the queue than the CPU can handle.

  Possible causes:

  - A process has created too many threads or child processes.
  - The CPU is overloaded.
- LoadAverageHigh CE S5
  ▼ Average load on node NODE_NAME is too high.

  Over the last 30 minutes, the average load on node `NODE_NAME` has been higher than or equal to XXX per core.

  There are more processes in the queue than the CPU can handle.

  Possible causes:

  - A process has created too many threads or child processes.
  - The CPU is overloaded.
- NodeDiskBytesUsage CE S5
  ▼ Node disk NODE_DISK_NAME on mount point MOUNTPOINT is using more than XXX% of its storage capacity.

  Deckhouse has detected that node disk `NODE_DISK_NAME` on mount point `MOUNTPOINT` is using more than XXX% of its storage capacity.

  Current storage usage: VALUE%

  Steps to resolve:

  1. Retrieve disk usage information on the node:

     ```
     ncdu -x MOUNTPOINT
     ```

  2. If the output shows high disk usage in the `/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/` directory, identify the pods with the highest usage:

     ```
     crictl stats -o json | jq '.stats[] | select((.writableLayer.usedBytes.value | tonumber) > 1073741824) | { meta: .attributes.labels, diskUsage: ((.writableLayer.usedBytes.value | tonumber) / 1073741824 * 100 | round / 100 | tostring + " GiB")}'
     ```

- NodeDiskBytesUsage CE S6
  ▼ Node disk NODE_DISK_NAME on mount point MOUNTPOINT is using more than XXX% of its storage capacity.

  Deckhouse has detected that node disk `NODE_DISK_NAME` on mount point `MOUNTPOINT` is using

more than XXX% of its storage capacity.

Current storage usage: VALUE%

Steps to resolve:

1. Retrieve disk usage information on the node:

   ```
   ncdu -X MOUNTPOINT
   ```

2. If the output shows high disk usage in the `/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/` directory, identify the pods with the highest usage:

   ```
   crictl stats -o json | jq '.stats[] | select((.writableLayer.usedBytes.value | tonumber) > 1073741824) | { meta: .attributes.labels, diskUsage: ((.writableLayer.usedBytes.value | tonumber) / 1073741824 * 100 | round / 100 | tostring + " GiB")}'
   ```

- NodeDiskInodesUsage CE S5
  ▾ Node disk NODE_DISK_NAME on mount point MOUNTPOINT is using more than XXX% of its storage capacity.

  Deckhouse has detected that node disk `NODE_DISK_NAME` on mount point `MOUNTPOINT` is using more than XXX% of its storage capacity.

  Current storage usage: VALUE%

- NodeDiskInodesUsage CE S6
  ▾ Node disk NODE_DISK_NAME on mount point MOUNTPOINT is using more than XXX% of its storage capacity.

  Deckhouse has detected that node disk `NODE_DISK_NAME` on mount point `MOUNTPOINT` is using more than XXX% of its storage capacity.

  Current storage usage: VALUE%

- PersistentVolumeClaimBytesUsage CE S4
  ▾ PersistentVolumeClaim NAMESPACE/PVC_NAME is using more than XXX% of the volume storage capacity.

  Deckhouse has detected that PersistentVolumeClaim `NAMESPACE/PVC_NAME` is using more than XXX% of the volume storage capacity.

  Current volume storage usage: VALUE%

  PersistentVolumeClaim `NAMESPACE/PVC_NAME` is used by the following pods:

  ```
  XXXXXX, XXXPOD_NAMEXXX
  ```

- PersistentVolumeClaimBytesUsage CE S5
  ▾ PersistentVolumeClaim NAMESPACE/PVC_NAME is using more than XXX% of the volume storage capacity.

  Deckhouse has detected that PersistentVolumeClaim `NAMESPACE/PVC_NAME` is using more than XXX% of the volume storage capacity.

  Currently volume storage usage: VALUE%

  PersistentVolumeClaim `NAMESPACE/PVC_NAME` is used by the following pods:

  ```
  XXXXXX, XXXPOD_NAMEXXX
  ```

- PersistentVolumeClaimInodesUsed CE S4
  ▾ PersistentVolumeClaim NAMESPACE/PVC_NAME is using more than XXX% of the volume inode capacity.

  Deckhouse has detected that PersistentVolumeClaim `NAMESPACE/PVC_NAME` is using more than XXX% of the volume inode capacity.

  Current volume inode usage: VALUE%

  PersistentVolumeClaim `NAMESPACE/PVC_NAME` is used by the following pods:

```
XXXXXX, XXXPOD_NAMEXXX
```

- PersistentVolumeClaimInodesUsed CE S5
  ▼ PersistentVolumeClaim NAMESPACE/PVC_NAME is using more than XXX% of the volume inode capacity.

  Deckhouse has detected that PersistentVolumeClaim `NAMESPACE/PVC_NAME` is using more than XXX% of the volume inode capacity.

  Current volume inode usage: VALUE%

  PersistentVolumeClaim `NAMESPACE/PVC_NAME` is used by the following pods:

  ```
  XXXXXX, XXXPOD_NAMEXXX
  ```

- StatefulSetAuthenticationFailure CE S7
  ▼ Unable to log in to the container registry using imagePullSecrets for the IMAGE_NAME image.

  Deckhouse was unable to log in to the container registry using `imagePullSecrets` for the `IMAGE_NAME` image.

  To resolve this issue, investigate the possible causes in the following sources:

  - The `NAMESPACE` namespace.
  - The StatefulSet `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- StatefulSetAuthorizationFailure CE S7
  ▼ Insufficient privileges to pull the IMAGE_NAME image using the specified imagePullSecrets.

  Deckhouse has insufficient privileges to pull the `IMAGE_NAME` image using the specified `imagePullSecrets`.

  To resolve this issue, investigate the possible causes in the following sources:

  - The `NAMESPACE` namespace.
  - The StatefulSet `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- StatefulSetBadImageFormat CE S7
  ▼ The IMAGE_NAME image name is incorrect.

  Deckhouse has detected that the `IMAGE_NAME` image name is incorrect.

  To resolve this issue, check that the `IMAGE_NAME` image name is spelled correctly in the following sources:

  - The `NAMESPACE` namespace.
  - The StatefulSet `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- StatefulSetImageAbsent CE S7
  ▼ The IMAGE_NAME image is missing from the registry.

  Deckhouse has detected that the `IMAGE_NAME` image is missing from the container registry.

  To resolve this issue, check whether the `IMAGE_NAME` image is available in the following sources:

  - The `NAMESPACE` namespace.
  - The StatefulSet `NAME`.
  - The `CONTAINER_NAME` container in the registry.
- StatefulSetImageAvailabilityUnknownError CE S7
  ▼ An unknown error occurred with the IMAGE_NAME image availability check.

  Deckhouse has detected an unknown error while checking `IMAGE_NAME` image availability in the following sources:

  - The `NAMESPACE` namespace.
  - The StatefulSet `NAME`.
  - The `CONTAINER_NAME` container in the registry.

  Review the image-availability-exporter logs to see the detailed error information:

  ```
  d8 k -n d8-monitoring logs -l app=image-availability-exporter -c image-availability-exporter
  ```

- StatefulSetRegistryUnavailable CE S7

▾ The container registry is not available for the IMAGE_NAME image.

Deckhouse has detected that the container registry is not available for the `IMAGE_NAME` image.

To resolve this issue, investigate the possible causes in the following sources:

- The `NAMESPACE` namespace.
- The StatefulSet `NAME`.
- The `CONTAINER_NAME` container in the registry.

Module ingress-nginx

- D8NginxIngressKruiseControllerPodIsRestartingTooOften CE S8
  ▾ Too many Kruise controller restarts detected.

  VALUE Kruise controller restarts detected in the last hour.

  Excessive Kruise controller restarts indicate that something is wrong. Normally, it should be up and running all the time.

  Steps to resolve:

  1. Check events associated with `kruise-controller-manager` in the `d8-ingress-nginx` namespace. Look for issues related to node failures or memory shortages (OOM events):

     ```
     d8 k -n d8-ingress-nginx get events | grep kruise-controller-manager
     ```

  2. Analyze the controller's pod descriptions to identify restarted containers and possible causes. Pay attention to exit codes and other details:

     ```
     d8 k -n d8-ingress-nginx describe pod -lapp=kruise,control-plane=controller-manager
     ```

  3. In case the `kruise` container has restarted, get a list of relevant container logs to identify any meaningful errors:

     ```
     d8 k -n d8-ingress-nginx logs -lapp=kruise,control-plane=controller-manager -c kruise
     ```

- GeoIPDownloadErrorDetectedFromMaxMind CE S4
  ▾ GeoIP DB download error in JOB_NAME (pod POD_NAME), edition XXX.

  The component `JOB_NAME` (pod `POD_NAME`) failed to download the GeoIP database from MaxMind for edition `XXX`. If this issue persists, investigate network connectivity or database availability.

- NginxIngressConfigTestFailed CE S4
  ▾ Configuration test failed on Ingress NGINX CONTROLLER_NAMESPACE/CONTROLLER_NAME.

  The configuration test (`nginx -t`) for the `CONTROLLER_NAME` Ingress controller in the `CONTROLLER_NAMESPACE` namespace has failed.

  Steps to resolve:

  1. Check the controller logs:

     ```
     d8 k -n CONTROLLER_NAMESPACE logs CONTROLLER_POD_NAME -c controller
     ```

  2. Find the most recently created Ingress in the cluster:

     ```
     d8 k get ingress --all-namespaces --sort-by="metadata.creationTimestamp"
     ```

  3. Check for errors in the `configuration-snippet` or `server-snippet` annotations.

- NginxIngressDaemonSetNotUpToDate CE S9
  ▾ There were VALUE outdated pods in Ingress NGINX DaemonSet NAMESPACE/DAEMONSET_NAME over the last 20 minutes.

  Deckhouse has detected VALUE outdated pods in Ingress NGINX DaemonSet `NAMESPACE/DAEMONSET_NAME` over the last 20 minutes.

  Steps to resolve:

1. Check the DaemonSet's status:

   ```
   d8 k -n NAMESPACE get ads DAEMONSET_NAME
   ```

2. Analyze the DaemonSet's description:

   ```
   d8 k -n NAMESPACE describe ads DAEMONSET_NAME
   ```

3. If the parameter `Number of Nodes Scheduled with Up-to-date Pods` does not match `Current Number of Nodes Scheduled`, check the 'nodeSelector' and 'toleration' settings of the corresponding Ingress NGINX Controller and compare them to the 'labels' and 'taints' settings of the relevant nodes.

- NginxIngressDaemonSetReplicasUnavailable CE S4
  ▾ No available replicas remaining in Ingress NGINX DaemonSet NAMESPACE/DAEMONSET_NAME.

  Deckhouse has detected that there are no available replicas remaining in Ingress NGINX DaemonSet `NAMESPACE/DAEMONSET_NAME`.

  List of unavailable pods:

  ```
  XXXXXX, XXXPOD_NAMEXXX
  ```

  If you know where the DaemonSet should be scheduled, run the command below to identify the problematic nodes. Use a label selector for pods, if needed.

  ```
  d8 k -n NAMESPACE get pod -ojson | jq -r '.items[] |
  select(.metadata.ownerReferences[] | select(.name =="DAEMONSET_NAME")) |
  select(.status.phase != "Running" or ([ .status.conditions[] | select(.type ==
  "Ready" and .status == "False") ] | length ) == 1 ) |
  .spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.node
  SelectorTerms[].matchFields[].values[]'
  ```

- NginxIngressDaemonSetReplicasUnavailable CE S6
  ▾ Some replicas of Ingress NGINX DaemonSet NAMESPACE/DAEMONSET_NAME are unavailable.

  Deckhouse has detected that some replicas of Ingress NGINX DaemonSet `NAMESPACE/DAEMONSET_NAME` are unavailable.

  Current number: VALUE unavailable replica(s).

  List of unavailable pods:

  ```
  XXXXXX, XXXPOD_NAMEXXX
  ```

  If you know where the DaemonSet should be scheduled, run the command below to identify the problematic nodes. Use a label selector for pods, if needed.

  ```
  d8 k -n NAMESPACE get pod -ojson | jq -r '.items[] |
  select(.metadata.ownerReferences[] | select(.name =="DAEMONSET_NAME")) |
  select(.status.phase != "Running" or ([ .status.conditions[] | select(.type ==
  "Ready" and .status == "False") ] | length ) == 1 ) |
  .spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.node
  SelectorTerms[].matchFields[].values[]'
  ```

- NginxIngressMaxMindAccountIDNotSet CE S6
  ▾ MaxMind Account ID is missing for the Ingress NGINX Controller "XXX".

  The controller **"XXX"** downloads the GeoIP database using the legacy flow because `spec.geoIP2.maxmindAccountID` is not set. This fallback is less reliable and may be throttled by MaxMind. Specify both `maxmindAccountID` and `maxmindLicenseKey` in the corresponding `ingressnginxcontroller` resource to use the official GeoIP update client.

- NginxIngressPodIsRestartingTooOften CE S4
  ▾ Too many Ingress NGINX restarts detected.

  VALUE Ingress NGINX Controller restarts detected in the last hour

VALUE Ingress NGINX Controller restarts detected in the last hour.

Excessive Ingress NGINX restarts indicate that something is wrong. Normally, it should be up and running all the time.

- NginxIngressPodsReportConfigurationHashDrift CE S4
  ▼ **Ingress NGINX Controller pods report different configuration hashes.**

  The `CONTROLLER_NAME` Ingress NGINX Controller pods report different configuration hashes. It can result in inconsistent processing of incoming requests.

  To diagnose the issue, follow these steps:

  1. Make sure that the pods of `CONTROLLER_NAME` Ingress NGINX Controller are not being updated at the moment. During an update, pods of a single controller may report different config hashes for some time.

     ```
     d8 k -n NAMESPACE get pod -l app=controller,name=CONTROLLER_NAME
     ```

  2. Check the associated metrics `nginx ingress controller_config_hash{namespace="d8-ingress-nginx", controller="CONTROLLER_NAME"}` manually to find the outliers.

  3. Check the controller logs for any configuration errors resulting in the configuration drift across the pods:

     ```
     d8 k -n NAMESPACE logs -l app=controller,name=CONTROLLER_NAME -c controller --tail=10000
     ```

- NginxIngressProfilingIsEnabled CE S4
  ▼ **Warning: Profiling mode is enabled in the Ingress NGINX Controller "XXX".**

  Profiling mode is enabled for the Ingress NGINX Controller **“XXX”**. This may increase memory consumption, slow down request processing, and the process may run as root. It is recommended to disable profiling unless it is actively needed for debugging or performance analysis. To disable profiling, set `nginxProfilingEnabled: false` in the `ingressnginxcontroller` resource configuration for this controller.

- NginxIngressProtobufExporterHasErrors CE S8
  ▼ **The Ingress NGINX sidecar container with protobuf_exporter has ERROR_TYPE errors.**

  Deckhouse has detected that the Ingress NGINX sidecar container with `protobuf_exporter` has ERROR_TYPE errors.

  To resolve the issue, check the Ingress controller’s logs:

  ```
  d8 k -n d8-ingress-nginx logs $(d8 k -n d8-ingress-nginx get pods -l
  app=controller,name=CONTROLLER_NAME -o wide | grep NODE_NAME | awk '{print $1}') -c
  protobuf-exporter
  ```

- NginxIngressSslExpired CE S4
  ▼ **Certificate has expired.**

  The SSL certificate for HOST_NAME in the `NAMESPACE` namespace has expired.

  To verify the certificate, run the following command:

  ```
  d8 k -n NAMESPACE get secret SECRET_NAME -o json | jq -r '.data."tls.crt" | @base64d' |
  openssl x509 -noout -alias -subject -issuer -dates
  ```

  The site at `https://HOST_NAME` is not accessible.

- NginxIngressSslWillExpire CE S5
  ▼ **Certificate is expiring soon.**

  The SSL certificate for HOST_NAME in the `NAMESPACE` namespace will expire in less than two weeks.

  To verify the certificate, run the following command:

  ```
  d8 k -n NAMESPACE get secret SECRET_NAME -o json | jq -r '.data."tls.crt" | @base64d' |
  openssl x509 -noout -alias -subject -issuer -dates
  ```

- NginxIngressValidationIsDisabled CE S4

▾ Warning: Ingress resource validation in the Ingress NGINX Controller is currently disabled.

Validation is disabled to reduce load on the master nodes, as it requires additional resources. To re-enable validation, remove the annotation `network.deckhouse.io/ingress-nginx-validation-suspended` from the `ingressnginxcontroller` resource.

To find which `IngressNginxController` resources have the annotation, use the following command:

```
d8 k get ingressnginxcontrollers.deckhouse.io -o json | jq -r '.items[] |
select(.metadata.annotations."network.deckhouse.io/ingress-nginx-validation-
suspended" != null) | "\(.metadata.name)"'
```

Module istio

- D8IstioActualDataPlaneVersionNotEqualDesired EE S8
  ▾ There are pods with Istio data plane version VERSION_NUMBER, but desired version is XXX.

  There are pods in the `NAMESPACE` namespace with Istio data plane version `VERSION_NUMBER`, while the desired version is `XXX`. As a result, the Istio version will be changed after the pod is restarted.

  To resolve the issue, use the following cheat sheet:

  ```
  ### Namespace-wide configuration
  # `istio.io/rev=vXYZ`: Use a specific revision.
  # `istio-injection=enabled`: Use the global revision.
  d8 k get ns NAMESPACE --show-labels

  ### Pod-wide configuration
  d8 k -n NAMESPACE get pods -l istio.io/rev=DESIRED_VISION
  ```

- D8IstioActualVersionIsNotInstalled EE S4
  ▾ The control plane version for pods with injected sidecars isn't installed.

  There are pods in the `NAMESPACE` namespace with injected sidecars of version `VERSION_NUMBER` (revision `REVISION_NUMBER`), but the corresponding control plane version is not installed. As a result, these pods have lost synchronization with the state in Kubernetes.

  To resolve this issue, install the required control plane version. Alternatively, update the namespace or pod configuration to match an installed control plane version.

  To identify orphaned pods, run the following command:

  ```
  d8 k -n NAMESPACE get pods -l 'service.istio.io/canonical-name' -o json | jq --arg
  revision REVISION_NUMBER '.items[] |
  select(.metadata.annotations."sidecar.istio.io/status" // "{}" | fromjson |
  .revision == $revision) | .metadata.name'
  ```

- D8IstioAdditionalControlplaneDoesntWork CE S4
  ▾ Additional control plane isn't working.

  Deckhouse has detected that the additional Istio control plane `ISTIO_REVISION_LABEL` isn’t working.

  As a result, sidecar injection for pods with `ISTIO_REVISION_LABEL` isn’t working as well.

  To check the status of the control plane pods, run the following command:

  ```
  d8 k get pods -n d8-istio -l istio.io/rev=ISTIO_REVISION_LABEL
  ```

- D8IstioDataPlaneVersionMismatch EE S8
  ▾ There are pods with data plane version different from the control plane version.

  There are pods in the `NAMESPACE` namespace with Istio data plane version `VERSION_NUMBER`, which is different from the control plane version `DESIRED_VERSION`.

  Steps to resolve the issue:

  1. Restart affected pods and use the following PromQL query to get a full list:

```
max by (namespace, dataplane_pod)
(d8_istio_dataplane_metadata{full_version="VERSION_NUMBER"})
```

2. Use the automatic Istio data plane upgrade described in the guide.

- D8IstioDataPlaneWithoutIstioInjectionConfigured EE S4
  ▼ Detected pods with Istio sidecars but istio-injection isn't configured.

  There are pods in the `NAMESPACE` namespace with Istio sidecars, but istio-injection isn’t configured. As a result, these pods will lose their Istio sidecars after being recreated.

  To identify the affected pods, run the following command:

  ```
  d8 k -n NAMESPACE get pods -o json | jq -r --arg revision REVISION_NUMBER '.items[] |
  select(.metadata.annotations."sidecar.istio.io/status" // "{}" | fromjson |
  .revision == $revision) | .metadata.name'
  ```

- D8IstioDeprecatedIstioVersionInstalled CE
  ▼ The installed Istio version has been deprecated.

  Deckhouse has detected that a deprecated Istio version `VERSION_NUMBER` is installed.

  Support for this version will be removed in upcoming Deckhouse releases. The higher the alert severity, the greater the probability of support being discontinued.

  To learn how to upgrade Istio, refer to the upgrade guide.

- D8IstioDesiredVersionIsNotInstalled EE S6
  ▼ Desired control plane version isn't installed.

  There is a desired Istio control plane version `XXX` (revision `REVISION_NUMBER`) configured for pods in the `NAMESPACE` namespace, but that version isn’t installed. As a result, pods can’t be recreated in the `NAMESPACE` namespace.

  To resolve this issue, install the desired control plane version. Alternatively, update the namespace or pod configuration to match an installed control plane version.

  Use the following cheat sheet:

  ```
  ### Namespace-wide configuration
  # `istio.io/rev=vXYZ`: Use a specific revision.
  # `istio-injection=enabled`: Use the global revision.
  d8 k get ns NAMESPACE --show-labels

  ### Pod-wide configuration
  d8 k -n NAMESPACE get pods -l istio.io/rev=REVISION_NUMBER
  ```

- D8IstioFederationMetadataEndpointDoesntWork EE S6
  ▼ Federation metadata endpoint failed.

  The metadata endpoint `ENDPOINT_NAME` for IstioFederation `FEDERATION_NAME` has failed to fetch via the Deckhouse hook.

  To reproduce the request to the public endpoint, run the following command:

  ```
  curl ENDPOINT_NAME
  ```

  To reproduce the request to private endpoints (run from the Deckhouse pod), run the following:

  ```
  KEY="$(deckhouse-controller module values istio -o json | jq -r
  .internal.remoteAuthnKeypair.priv)"
  LOCAL_CLUSTER_UUID="$(deckhouse-controller module values -g istio -o json | jq -r
  .global.discovery.clusterUUID)"
  REMOTE_CLUSTER_UUID="$(d8 k get istiofederation FEDERATION_NAME -o json | jq -r
  .status.metadataCache.public.clusterUUID)"
  TOKEN="$(deckhouse-controller helper gen-jwt --private-key-path <(echo "$KEY") --claim
  iss=d8-istio --claim sub=$LOCAL_CLUSTER_UUID --claim aud=$REMOTE_CLUSTER_UUID --
  claim scope=private-federation --ttl 1h)"
  curl -H "Authorization: Bearer $TOKEN" ENDPOINT_NAME
  ```

- D8IstioGlobalControlplaneDoesntWork CE S4
  ▼ **Global control plane isn't working.**

  Deckhouse has detected that the global Istio control plane `ISTIO_REVISION_LABEL` isn’t working.

  As a result, sidecar injection for pods with global revision isn’t working as well, and the validating webhook for Istio resources is absent.

  To check the status of the control plane pods, run the following command:

  ```
  d8 k get pods -n d8-istio -l istio.io/rev=ISTIO_REVISION_LABEL
  ```

- D8IstioMulticlusterMetadataEndpointDoesntWork EE S6
  ▼ **Multicluster metadata endpoint failed.**

  The metadata endpoint `ENDPOINT_NAME` for IstioMulticluster `MULTICLUSTER_NAME` has failed to fetch via the Deckhouse hook.

  To reproduce the request to the public endpoint, run the following command:

  ```
  curl ENDPOINT_NAME
  ```

  To reproduce the request to private endpoints (run from the `d8-system/deckhouse` pod), run the following:

  ```
  KEY="$(deckhouse-controller module values istio -o json | jq -r
  .internal.remoteAuthnKeypair.priv)"
  LOCAL_CLUSTER_UUID="$(deckhouse-controller module values -g istio -o json | jq -r
  .global.discovery.clusterUUID)"
  REMOTE_CLUSTER_UUID="$(d8 k get istiomulticluster MULTICLUSTER_NAME -o json | jq -r
  .status.metadataCache.public.clusterUUID)"
  TOKEN="$(deckhouse-controller helper gen-jwt --private-key-path <(echo "$KEY") --claim
  iss=d8-istio --claim sub=$LOCAL_CLUSTER_UUID --claim aud=$REMOTE_CLUSTER_UUID --
  claim scope=private-multicluster --ttl 1h)"
  curl -H "Authorization: Bearer $TOKEN" ENDPOINT_NAME
  ```

- D8IstioMulticlusterRemoteAPIHostDoesntWork EE S6
  ▼ **Multicluster remote API host health check failed.**

  The remote API host `API_HOST` for IstioMulticluster `MULTICLUSTER_NAME` has failed the health check performed by the Deckhouse monitoring hook.

  To reproduce the request (run from the `d8-system/deckhouse` pod), run the following:

  ```
  TOKEN="$(deckhouse-controller module values istio -o json | jq -r --arg ah API_HOST
  '.internal.multiclusters[]| select(.apiHost == $ah)| .apiJWT ')"
  curl -H "Authorization: Bearer $TOKEN" https://API_HOST/version
  ```

- D8IstioOperatorReconcileError CE S5
  ▼ **The istio-operator is unable to reconcile Istio control plane setup.**

  Deckhouse has detected an error in the `istio-operator` reconciliation loop.

  To investigate the issue, check the operator logs:

  ```
  d8 k -n d8-istio logs -l app=operator,revision=REVISION_NUMBER
  ```

- D8IstioPodsWithoutIstioSidecar EE S4
  ▼ **Detected pods without Istio sidecars but with istio-injection configured.**

  There is a pod `POD_NAME` in the `NAMESPACE` namespace without Istio sidecars, but with istio-injection configured.

  To identify the affected pods, run the following command:

  ```
  d8 k -n NAMESPACE get pods -l '!service.istio.io/canonical-name' -o json | jq -r
  '.items[] | select(.metadata.annotations."sidecar.istio.io/inject" != "false")
  | .metadata.name'
  ```

- D8IstioRemoteClusterNotSynced EE S4
  ▼ **Istio remote cluster CLUSTER_ID is not synchronized.**

  The Istio control plane instance `INSTANCE_NAME` cannot synchronize with the remote cluster `CLUSTER_ID`.

  Possible causes:

  - The remote cluster is offline.
  - The remote API endpoint is not reachable.
  - The remote ServiceAccount token is invalid or expired.
  - There is a TLS or certificate issue between the clusters.
  - Remote and local Istio versions mismatch.
- D8IstioReservedUIDConflict CE S4
  ▼ **Application container is running as UID 1337 (reserved by Istio).**

  Container `CONTAINER_NAME` in pod `POD_NAME` in namespace `NAMESPACE` is running as UID `1337`.

  UID `1337` is reserved by Istio for the `istio-proxy` sidecar container. When application containers run with this UID, their traffic bypasses Istio completely — no routing rules, mTLS, or telemetry will be applied.

  To fix the issue, change the `runAsUser` in the container's or pod's `securityContext` to a different UID.

- D8IstioVersionIsIncompatibleWithK8sVersion CE S3
  ▼ **The installed Istio version is incompatible with the Kubernetes version.**

  The installed Istio version `VERSION_NUMBER` may not work properly with the current Kubernetes version `VERSION_NUMBER` because it's not supported officially.

  To resolve the issue, upgrade Istio following the guide.

- IstioIrrelevantExternalServiceFound CE S5
  ▼ **External service found with irrelevant ports specifications.**

  A service `NAME` in the `NAMESPACE` namespace has an irrelevant port specification.

  The `.spec.ports[]` field isn't applicable for services of the `ExternalName` type. However, Istio renders port listeners for external services as `0.0.0.0:port`, which captures all traffic to the specified port. This can cause problems for services that aren't registered in the Istio registry.

  To resolve the issue, remove the `.spec.ports` section from the service configuration. It is safe.

Module kube-dns

- KubernetesCoreDNSHasCriticalErrors CE S5
  ▼ **Critical errors found in CoreDNS.**

  Deckhouse has detected at least one critical error in the CoreDNS pod POD_NAME.

  To resolve the issue, review the container logs:

  ```
  d8 k -n kube-system logs POD_NAME
  ```

Module log-shipper

- D8LogShipperAgentNotScheduledInCluster CE S7
  ▼ **The log-shipper-agent pods can't be scheduled in the cluster.**

  Deckhouse has detected that a number of `log-shipper-agent` pods are not scheduled.

  To resolve this issue, do the following:

  1. Check the state of the `d8-log-shipper/log-shipper-agent` DaemonSet:

     ```
     d8 k -n d8-log-shipper get daemonsets --selector=app=log-shipper
     ```

  2. Check the state of the `d8-log-shipper/log-shipper-agent` pods:

     ```
     d8 k -n d8-log-shipper get pods --selector=app=log-shipper-agent
     ```

  3. If you know where the DaemonSet should be scheduled, run the following command to

identify the problematic nodes:

```
d8 k -n d8-log-shipper get pod -ojson | jq -r '.items[] |
select(.metadata.ownerReferences[] | select(.name =="log-shipper-agent")) |
select(.status.phase != "Running" or ([ .status.conditions[] | select(.type
== "Ready" and .status == "False") ] | length ) == 1 ) |
.spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.
nodeSelectorTerms[].matchFields[].values[]'
```

- D8LogShipperClusterLogDestinationD8LokiAuthorizationRequired CE S9
  ▼ Authorization parameters required for the ClusterLogDestination resource.

  Deckhouse has detected the ClusterLogDestination resource `RESOURCE_NAME` without authorization parameters.

  Add the authorization parameters to the ClusterLogDestination resource following the instructions.

- D8LogShipperCollectLogErrors CE S4
  ▼ The log-shipper-agent pods can't collect logs to COMPONENT_ID on the NODE_NAME node.

  Deckhouse has detected that the HOST_NAME `log-shipper-agent` on the `NODE_NAME` node has failed to collect metrics for more than 10 minutes.

  This is caused by the `ERROR_TYPE` errors occurred during the `STAGE_NAME` stage while reading `COMPONENT_TYPE`.

  To resolve this, check the pod logs or follow advanced instructions:

  ```
  d8 k -n d8-log-shipper logs HOST_NAME` -c vector
  ```

- D8LogShipperConfigInvalid CE S4
  ▼ Vector configuration is invalid in the cluster.

  At least one Vector DaemonSet Pod is reporting an invalid configuration.

  To resolve this, check the container logs or follow advanced instructions:

  ```
  d8 k -n d8-log-shipper logs HOST_NAME -c vector-reloader
  ```

- D8LogShipperDestinationErrors CE S4
  ▼ The log-shipper-agent pods can't send logs to COMPONENT_ID on the NODE_NAME node.

  Deckhouse has detected that the HOST_NAME `log-shipper-agent` on the NODE_NAME node has failed to send a log for more than 10 minutes.

  This is caused by the `ERROR_TYPE` errors occurred during the `STAGE_NAME` stage while sending logs to `COMPONENT_TYPE`.

  To resolve this, check the pod logs or follow advanced instructions:

  ```
  d8 k -n d8-log-shipper logs HOST_NAME -c vector
  ```

- D8LogShipperLogsDroppedByRateLimit CE S4
  ▼ The log-shipper-agent pods are dropping logs to COMPONENT_ID on the NODE_NAME node.

  Rate-limiting rules have been applied, and the `log-shipper-agent` on the `NODE_NAME` node has been dropping logs for more than 10 minutes.

  To resolve this, check the pod logs or follow advanced instructions:

  ```
  d8 k -n d8-log-shipper get pods -o wide | grep NODE_NAME
  ```

Module loki

- LokiDiscardedSamplesWarning CE S4
  ▼ Loki is discarding samples during ingestion.

  Loki is discarding samples during ingestion because they fail validation. Samples are being

discarded because of `XXX` at a rate of XXX samples per second. For more information, refer to the `loki` FAQ.

- LokiInsufficientDiskForRetention CE S4
  ▼ **Not enough disk space to retain logs for 168 hours**

  Not enough disk space to retain logs for 168 hours. Current effective retention period is VALUE hours.

  You need either decrease expected `retentionPeriodHours` or increase resize Loki PersistentVolumeClaim

Module metallb

- D8MetallbBGPSessionDown EE S4
  ▼ **MetalLB BGP session is down.**

  JOB_NAME, MetalLB CONTAINER_NAME on POD_NAME has BGP session PEER down.

  Check the logs for details:

  ```
  d8 k -n d8-metallb logs daemonset/speaker -c speaker
  ```

- D8MetallbConfigNotLoaded EE S4
  ▼ **The MetalLB configuration hasn't been loaded.**

  JOB_NAME, MetalLB CONTAINER_NAME on POD_NAME hasn’t been loaded.

  To find the cause of the issue, review the controller logs:

  ```
  d8 k -n d8-metallb logs deploy/controller -c controller
  ```

- D8MetallbConfigStale EE S4
  ▼ **MetalLB is running on a stale configuration.**

  JOB_NAME, MetalLB CONTAINER_NAME on POD_NAME is running on a stale configuration because the latest configuration failed to load.

  To find the cause of the issue, review the controller logs:

  ```
  d8 k -n d8-metallb logs deploy/controller -c controller
  ```

- D8MetallbNotSupportedServiceAnnotationsDetected SE S4
  ▼ **The annotation 'ANNOTATION_NAME' has been deprecated for the service 'NAME' in the 'NAMESPACE' namespace.**

  The annotation ‘ANNOTATION_NAME’ has been deprecated for the service ‘NAME’ in the ‘NAMESPACE’ namespace.

  The following service annotations are no longer effective:

  - `metallb.universe.tf/ip-allocated-from-pool`: Remove this annotation.
  - `metallb.universe.tf/address-pool`: Replace it with the `.spec.loadBalancerClass` parameter or use the `network.deckhouse.io/metal-load-balancer-class` annotation, referencing the appropriate MetalLoadBalancerClass.
  - `metallb.universe.tf/loadBalancerIPs`: Replace it with `network.deckhouse.io/load-balancer-ips: IP`.
  - `metallb.universe.tf/allow-shared-ip`: Replace it with `network.deckhouse.io/load-balancer-shared-ip-key`.
  - `metallb.io/ip-allocated-from-pool`: Remove this annotation.
  - `metallb.io/address-pool`: Replace it with the `.spec.loadBalancerClass` parameter or use the `network.deckhouse.io/metal-load-balancer-class` annotation, referencing the appropriate MetalLoadBalancerClass.
  - `metallb.io/loadBalancerIPs`: Replace it with `network.deckhouse.io/load-balancer-ips: IP`.
  - `metallb.io/allow-shared-ip`: Replace it with `network.deckhouse.io/load-balancer-shared-ip-key`.

  Please note. Existing LoadBalancer services of Deckhouse have been migrated automatically, but the new ones will not be.

- D8MetallbObsoleteLayer2PoolsAreUsed SE S7
  ▼ **The metallb module has obsolete layer2 pools configured.**

In ModuleConfig version 2, addressPool ‘NAME’ of type “layer2” are ignored. They should be removed from the configuration.

- D8MetallbUpdateMCVersionRequired SE S5
  ▼ **The metallb ModuleConfig settings are outdated.**

  D8 MetalLB settings are outdated.

  To resolve this issue, increase version in the ModuleConfig `metallb`.

- L2LoadBalancerOrphanServiceFound SE S4
  ▼ **Orphaned service with an irrelevant L2LoadBalancer name has been found.**

  The cluster contains an orphaned service `NAME` in the `NAMESPACE` namespace with an irrelevant L2LoadBalancer name.

  To resolve this issue, verify the L2LoadBalancer name specified in the annotation `network.deckhouse.io/l2-load-balancer-name`.

Module monitoring-applications

- D8OldPrometheusTargetFormat FE S6
  ▼ **Services with the prometheus-target label are being used for metric collection.**

  Deckhouse has detected that services with the `prometheus-target` label are being used to collect metrics in the cluster.

  The label format has been changed. To resolve the issue, replace the `prometheus-target` label with `prometheus.deckhouse.io/target`.

  To list all services labeled with `prometheus-target`, run the following command:

  ```
  d8 k get service --all-namespaces --show-labels | grep prometheus-target
  ```

Module monitoring-custom

- CustomPodMonitorFoundInCluster CE S9
  ▼ **Deckhouse namespace contains PodMonitors not created by Deckhouse.**

  There are PodMonitors in the Deckhouse namespace that were not created by Deckhouse.

  To resolve the issue, move these PodMonitors to the `user-spec` namespace by removing the `heritage: deckhouse` label.

  To list all PodMonitors in the Deckhouse namespace, run the following command:

  ```
  d8 k get podmonitors --all-namespaces -l heritage!=deckhouse
  ```

  For more information on metric collection, refer to the Prometheus module FAQ.

- CustomServiceMonitorFoundInD8Namespace CE S9
  ▼ **Deckhouse namespace contains ServiceMonitors not created by Deckhouse.**

  There are ServiceMonitors in the Deckhouse namespace that were not created by Deckhouse.

  To resolve the issue, move these ServiceMonitors to the `user-spec` namespace by removing the `heritage: deckhouse` label.

  To list all ServiceMonitors in the Deckhouse namespace, run the following command:

  ```
  d8 k get servicemonitors --all-namespaces -l heritage!=deckhouse
  ```

  For more information on metric collection, refer to the Prometheus module FAQ.

- D8CustomPrometheusRuleFoundInCluster CE S9
  ▼ **Deckhouse namespace contains PrometheusRules not created by Deckhouse.**

  There are PrometheusRules in the Deckhouse namespace that were not created by Deckhouse.

  To resolve the issue, replace these PrometheusRules with the CustomPrometheusRules object.

  To list all PrometheusRules in the Deckhouse namespace, run the following command:

```
d8 k get prometheusrules --all-namespaces -l heritage!=deckhouse
```

For details on adding alerts and recording rules, refer to the Prometheus module FAQ.

- D8OldPrometheusCustomTargetFormat CE S9
  ▼ Services with the prometheus-custom-target label are being used for metric collection.

  Deckhouse has detected that services with the `prometheus-custom-target` label are being used to collect metrics in the cluster.

  The label format has been changed. To resolve the issue, replace the `prometheus-custom-target` label with `prometheus.deckhouse.io/custom-target`.

  To list all services labeled with `prometheus-custom-target`, run the following command:

  ```
  d8 k get service --all-namespaces --show-labels | grep prometheus-custom-target
  ```

  For more information on metric collection, refer to the Prometheus module FAQ.

- D8ReservedNodeLabelOrTaintFound CE S6
  ▼ Node NAME is using a reserved label or taint.

  Deckhouse has detected that node NAME is using one of the following:

  - A reserved `metadata.labels` object `node-role.deckhouse.io/`, which doesn't end with `(system|frontend|monitoring| deckhouse module name_)`.
  - A reserved `spec.taints` object `dedicated.deckhouse.io`, with a value other than `(system|frontend|monitoring|_deckhouse_module_name_)`.

  This alert may fire if the corresponding module is missing or disabled in the cluster. It can be safely ignored if you are preparing nodes to enable a module that is currently disabled.

  For instructions on how to resolve this issue, refer to the node allocation guide.

Module monitoring-deckhouse

- D8CNIEnabledMoreThanOne CE S2
  ▼ More than one CNI is enabled in the cluster.

  Deckhouse has detected that multiple CNIs are enabled in the cluster. For the cluster to work correctly, only one CNI must be enabled.

  To resolve this issue, disable any unnecessary CNI.

- D8CNIMisconfigured CE S5
  ▼ The parameters specified in ModuleConfig of CNI_NAME do not match the ones that are actually being used in the cluster.

  This happened because there were several sources for configuring CNI parameters in the cluster, and ModuleConfig did not have the highest priority previously.

  To resolve this issue, the following steps should be taken:

  1. Find the ConfigMap `d8-system/desired-cni-moduleconfig` in the cluster, which contains the actual ModuleConfig settings.

     ```
     d8 k -n d8-system get configmap desired-cni-moduleconfig -o yaml
     ```

  2. Apply this prepared ModuleConfig to the cluster. This will not cause any actual reconfigurations in the cluster and is completely safe.
  3. Once the parameters in "ModuleConfig" match those used in the cluster:
     - this alert will be resolved,
     - and the priority of the sources for configuring CNI will change, and ModuleConfig will become the main source of truth.
  4. After that, you can add the desired parameters to the ModuleConfig CNI and they will be applied immediately.
- D8DeckhouseConfigInvalid CE S5
  ▼ Deckhouse configuration is invalid.

  Deckhouse configuration contains errors.

Steps to troubleshoot:

1. Check Deckhouse logs by running the following command:

   ```
   d8 k -n d8-system logs -f -l app=deckhouse
   ```

2. Edit the Deckhouse global configuration:

   ```
   d8 k edit mc global
   ```

   Or edit configuration of a specific module:

   ```
   d8 k edit mc MODULE_NAME
   ```

- D8DeckhouseCouldNotDeleteModule CE S4
  ▼ **Deckhouse is unable to delete the MODULE_NAME module.**

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseCouldNotDiscoverModules CE S4
  ▼ **Deckhouse is unable to discover modules.**

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseCouldNotRunGlobalHook CE S5
  ▼ **Deckhouse is unable to run the HOOK_NAME global hook.**

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseCouldNotRunModule CE S4
  ▼ **Deckhouse is unable to start the MODULE_NAME module.**

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseCouldNotRunModuleHook CE S7
  ▼ **Deckhouse is unable to run the MODULE_NAME/HOOK_NAME module hook.**

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseCustomTargetDown CE S4

  Prometheus is unable to scrape custom metrics generated by Deckhouse hooks.

- D8DeckhouseDeprecatedConfigmapManagedByArgoCD CE S4
  ▼ **Deprecated Deckhouse ConfigMap managed by Argo CD.**

  The Deckhouse ConfigMap is no longer used.

  To resolve this issue, remove the `d8-system/deckhouse` ConfigMap from Argo CD.

- D8DeckhouseGlobalHookFailsTooOften CE S9
  ▼ **The HOOK_NAME Deckhouse global hook is crashing too frequently.**

  The `HOOK_NAME` hook has failed multiple times in the last `__SCRAPE_INTERVAL_X_4__`.

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseHasNoAccessToRegistry CE S7
  ▼ **Deckhouse is unable to connect to the registry.**

  Deckhouse can’t connect to the registry (typically `registry.deckhouse.io`) to check for a new Docker image. These checks are performed every 15 seconds. Without access to the registry, automatic updates are unavailable.

  This alert often indicates that the Deckhouse Pod is experiencing connectivity issues with the Internet.

- D8DeckhouseIsHung CE S4
  ▼ **Deckhouse is down.**

  Deckhouse is probably down, since the `deckhouse_live_ticks` metric in Prometheus has stopped increasing. This metric is expected to increment every 10 seconds.

- D8DeckhouseIsNotOnReleaseChannel CE S9
  ▼ **Deckhouse isn't subscribed to any regular release channels.**

  Deckhouse in this cluster isn’t subscribed to any of the regular release channels: `Alpha`, `Beta`, `EarlyAccess`, `Stable`, or `RockSolid`.

  To resolve this issue, follow these steps:

  1. Check the current release channel used in the cluster:

     ```
     d8 k -n d8-system  get deploy deckhouse -o json | jq
     '.spec.template.spec.containers[0].image' -r
     ```

  2. Subscribe to one of the regular release channels by adjusting the `deckhouse` module configuration.

- D8DeckhouseModuleHookFailsTooOften CE S9
  ▼ **The MODULE_NAME/HOOK_NAME Deckhouse hook is crashing too frequently.**

  The `HOOK NAME` hook of the `MODULE_NAME` module has failed multiple times in the last `__SCRAPE_INTERVAL_X_4__`.

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseModuleUpdatePolicyNotFound CE S5
  ▼ **Module update policy not found for MODULE_RELEASE.**

  The module update policy for MODULE_RELEASE is missing.

  To resolve this issue, remove the label from the module release using the following command:

  ```
  d8 k label mr MODULE_RELEASE modules.deckhouse.io/update-policy-
  ```

  A new suitable policy will be detected automatically.

- D8DeckhouseModuleValidationError CE S5
  ▼ **Module configuration failed for module MODULE_NAME.**

  Initial config for module MODULE_NAME is not valid.

  You can get more details via

  ```
  d8 k get mr -l module=MODULE_NAME
  ```

  Provided error: XXX

- D8DeckhousePodIsNotReady CE S4

  The Deckhouse Pod is NOT Ready.

- D8DeckhousePodIsNotRunning CE S4

  The Deckhouse Pod is NOT Running.

- D8DeckhousePodIsRestartingTooOften CE S9
  ▼ **Excessive Deckhouse restarts detected.**

Number of restarts in the last hour: VALUE.

Excessive Deckhouse restarts indicate a potential issue. Normally, Deckhouse should be up and running continuously.

To investigate the issue, check the logs by running the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse
```

- D8DeckhouseQueueIsHung CE S7
  ▼ The QUEUE_NAME Deckhouse queue is stuck with VALUE pending task(s).

  Deckhouse cannot finish processing of the `QUEUE_NAME` queue, which currently has VALUE pending task(s).

  To investigate the issue, check the logs by running the following command:

  ```
  d8 k -n d8-system logs -f -l app=deckhouse
  ```

- D8DeckhouseSelfTargetAbsent CE S4

  There is no Deckhouse target in Prometheus.

- D8DeckhouseSelfTargetDown CE S4

  Prometheus is unable to scrape Deckhouse metrics.

- D8DeckhouseWatchErrorOccurred CE S5
  ▼ Possible API server connection error in the client-go informer.

  Deckhouse has detected an error in the client-go informer, possibly due to connection issues with the API server.

  Steps to investigate:

  1. Check Deckhouse logs for more information by running:

     ```
     d8 k -n d8-system logs deploy/deckhouse | grep error | grep -i watch
     ```

  2. This alert attempts to detect a correlation between the faulty snapshot invalidation and API server connection errors, specifically for the `handle-node-template` hook in the `node-manager` module.

     To compare the snapshot with the actual node objects for this hook, run the following command:

     ```
     diff -u <(d8 k get nodes -o jsonpath='{range .items[*]}{.metadata.name}{"\n"}
     {end}'|sort) <(d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-
     controller module snapshots node-manager -o json | jq '."040-node-
     manager/hooks/handle_node_templates.go"' | jq '.nodes.snapshot[] |
     .filterResult.Name' -r | sort)
     ```

- D8HasModuleConfigAllowedToDisable CE S4
  ▼ The ModuleConfig annotation for allowing module disabling is set.

  The ModuleConfig is pending module disabling.

  It is recommended that you keep your module configurations clean by removing unnecessary approval annotations.

  If you ignore this alert and do not clear the annotation, the module may be accidentally removed from the cluster, potentially leading to irreversible consequences.

  To resolve this issue and stop the alert, run the following command:

  ```
  d8 k annotate moduleconfig MODULE_NAME modules.deckhouse.io/allow-disabling-
  ```

- D8ModuleOutdatedByMajorVersion CE S9
  ▼ Module MODULE_NAME is running XXX major versions behind.

  Module `MODULE_NAME` is running XXX major versions behind on major version releases. Major version updates may contain breaking changes and require immediate attention and careful

planning.

To check the module releases status, run the following command:

```
d8 k get mr -l module=MODULE_NAME
```XXXXXX

This major release has from-to version constraints that must be met before applying the update.
To check the update instructions (from-to versions), run the following command:

```bash
d8 k get mr NAME -o jsonpath='{.spec.update.versions[*]}'
```

This will show you the update paths including the from-to versions defined in the release.XXX

To approve the module major release update, run the following command:

`bash d8 k annotate mr NAME modules.deckhouse.io/approved="true"` XXX

- D8NodeHasDeprecatedOSVersion CE S4
  ▼ **Nodes with deprecated OS versions detected.**

  Deckhouse has detected nodes running deprecated OS versions.

  Steps to troubleshoot:

  1. Get a list of affected nodes by running the following Prometheus query:

     ```
     kube_node_info{os_image=~"Ubuntu 18.04.*|Debian GNU/Linux 10.*|CentOS Linux 7.*"}
     ```

  2. Update the affected nodes to a supported OS version.

- DeckhouseEditionNotFound CE S6
  ▼ **Incorrect Deckhouse Kubernetes Platform edition.**

  Specify a Deckhouse Kubernetes Platform edition in `MODULE_NAME` ModuleConfig.

  ```
  d8 k patch moduleconfig MODULE_NAME --type merge --patch '{"spec": {"settings": {"license": {"edition": "DKP_EDITION"}}}}'
  ```

  Where `DKP_EDITION` is the edition you want to use: `CE`, `BE`, `EE`, `SE`, or `SE-plus`.

- DeckhouseHighMemoryUsage CE S3
  ▼ **Deckhouse memory usage is too high.**

  Deckhouse pod `POD_NAME` on node `NODE_NAME` has high memory usage.

  Please, if possible, get the dumps for debugging purposes (it will take around 30 seconds):

  ```
  curl -s "http://$(d8 k -n d8-system get pod POD_NAME -o jsonpath='{.status.hostIP}'):4222/debug/pprof/heap" -o /tmp/heap.pprof
  curl -s "http://$(d8 k -n d8-system get pod POD_NAME -o jsonpath='{.status.hostIP}'):4222/debug/pprof/goroutine" -o /tmp/goroutine.pprof
  curl -s "http://$(d8 k -n d8-system get pod POD_NAME -o jsonpath='{.status.hostIP}'):4222/debug/pprof/profile?seconds=30" -o /tmp/cpu.pprof
  ```

  and send these files (/tmp/heap.pprof, /tmp/cpu.pprof and /tmp/goroutine.pprof) to the support team.

- DeckhouseMigratedModuleNotDownloaded CE S6
  ▼ **Migrated module MODULE_NAME has not been downloaded yet.**

  Module `MODULE_NAME` specified in DeckhouseRelease was moved to an external source and is available in a ModuleSource registry, but it has not been downloaded yet (no deployed ModuleRelease exists). DeckhouseRelease will be locked until the module is downloaded.

  This alert is usually temporary while the module source controller downloads the module from the registry. If the alert persists, this may indicate:

  - The module's update policy or ModuleSource is misconfigured, preventing the module from being downloaded.
  - A network issue is preventing the module from being downloaded.

  To investigate the issue, do the following:

  - Check the ModuleRelease objects for the module:

    ```
    d8 k get mr -l module=MODULE_NAME
    ```

  - Check the configured ModuleSources:

    ```
    d8 k get modulesources
    ```

  - Check the `deckhouse` logs for more details:

    ```
    d8 k -n d8-system logs -f -l app=deckhouse
    ```

- DeckhouseMigratedModuleNotFound CE S6
  ▼ **Migrated module MODULE_NAME was not found in any ModuleSource registry.**

  Module `MODULE_NAME` specified in DeckhouseRelease was moved to an external source and was not found in any available ModuleSource registry. DeckhouseRelease will be locked until the module becomes available in one of the configured ModuleSource registries.

  This may indicate:

  - A ModuleSource registry is misconfigured.
  - The module has not been published to the registry yet.
  - A network issue is preventing access to the registry.

  To investigate the issue, do the following:

  - Check the configured ModuleSources:

    ```
    d8 k get modulesources
    ```

  - List the available module sources:

    ```
    d8 k -n d8-system exec -it deployment/deckhouse -c deckhouse -- deckhouse-controller
    registry get sources
    ```

  - List the modules available in a specific source:

    ```
    d8 k -n d8-system exec -it deployment/deckhouse -c deckhouse -- deckhouse-controller
    registry get modules source_name
    ```

  - Check the `deckhouse` logs for more details:

    ```
    d8 k -n d8-system logs -f -l app=deckhouse
    ```

- DeckhouseModuleUpdatingFailedBrokenSequence CE S4
  ▼ **Module update has failed. Updating sequence is broken.**

  The 'MODULE_NAME' Module update has failed.

  Current version: XXX. Desired version: VERSION_NUMBER.

  Attempt to download interim releases failed.

  Possible reasons:

  - The necessary versions of the Module image is not available in the registry.

  To resolve this issue, ensure that all minor versions of the 'MODULE_NAME' Module image between XXX and VERSION_NUMBER is available in the registry 'XXX'.

- DeckhouseModuleUpdatingFailedModuleIsNotValid CE S4
  ▼ **Module update has failed. Module is not valid.**

  The 'MODULE NAME' Module update has failed.

Desired version: VERSION_NUMBER.

Possible reasons:

- The Module image is corrupted.

To resolve this issue, ensure that VERSION_NUMBER image in the registry ‘XXX’ is valid .

To get the exact error, check Deckhouse logs: `d8 k -n d8-system logs deploy/deckhouse | grep "failed to download module"`

- DeckhouseReleaseDisruptionApprovalRequired CE S4
  ▼ Deckhouse release disruption approval required.

  The new Deckhouse release includes a disruptive update that requires manual approval.

  To check the details, run the following command:

  ```
  d8 k describe DeckhouseRelease NAME
  ```

  To approve the disruptive update, run the following command:

  ```
  d8 k annotate DeckhouseRelease NAME release.deckhouse.io/disruption-approved=true
  ```

- DeckhouseReleaseIsBlocked CE S5
  ▼ Deckhouse release requirements haven't been met.

  The requirements for the Deckhouse release haven’t been met.

  To check the details, run the following command:

  ```
  d8 k describe DeckhouseRelease NAME
  ```

- DeckhouseReleaseIsWaitingManualApproval CE S3
  ▼ A new Deckhouse release is awaiting manual approval.

  A new Deckhouse release is available but requires manual approval before it can be applied.

  To approve the release, run the following command:

  ```
  d8 k patch DeckhouseRelease NAME --type=merge -p='{"approved": true}'
  ```

- DeckhouseReleaseIsWaitingManualApproval CE S6
  ▼ A new Deckhouse release is awaiting manual approval.

  A new Deckhouse release is available but requires manual approval before it can be applied.

  To approve the release, run the following command:

  ```
  d8 k patch DeckhouseRelease NAME --type=merge -p='{"approved": true}'
  ```

- DeckhouseReleaseIsWaitingManualApproval CE S9
  ▼ A new Deckhouse release is awaiting manual approval.

  A new Deckhouse release is available but requires manual approval before it can be applied.

  To approve the release, run the following command:

  ```
  d8 k patch DeckhouseRelease NAME --type=merge -p='{"approved": true}'
  ```

- DeckhouseReleaseNotificationNotSent CE S4
  ▼ Deckhouse release notification webhook hasn't been sent.

  The Deckhouse release notification webhook failed to send.

  To check the notification webhook address, run the following command:

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

- DeckhouseUpdating CE S4

  Deckhouse is being updated to XXX.

Deckhouse is being updated to XXX.

- DeckhouseUpdatingFailed CE S4
  ▾ **Deckhouse update has failed.**

  The Deckhouse update has failed.

  Possible reasons:

  - The next minor or patch version of the Deckhouse image is not available in the registry.
  - The Deckhouse image is corrupted.

  Current version: VERSION_NUMBER.

  To resolve this issue, ensure that the next version of the Deckhouse image is available in the registry.

- ModuleAtConflict CE S4
  ▾ **Conflict detected for module MODULE_NAME.**

  Deckhouse has detected conflicting sources for the MODULE_NAME module.

  To resolve this issue, specify the correct source in the module configuration.

- ModuleConfigObsoleteVersion CE S4
  ▾ **ModuleConfig NAME is outdated.**

  Deckhouse has detected that ModuleConfig `NAME` is outdated.

  To resolve this issue, update ModuleConfig `NAME` to the latest version.

- ModuleHasDeprecatedUpdatePolicy CE S4
  ▾ **Module MODULE_NAME is matched by the deprecated update policy UPDATE_POLICY_NAME.**

  Deckhouse has detected that the module `MODULE_NAME` is using the deprecated update policy `UPDATE_POLICY_NAME`. The `v1alpha1` policy has a selector that no longer functions.

  To specify the update policy in the module configuration, run the following command:

  ```
  d8 k patch moduleconfig MODULE_NAME --type='json' -p='[{"op": "add", "path":
  "/spec/updatePolicy", "value": "UPDATE_POLICY_NAME"}]'
  ```

  After resolving all alerts related to the update policy `UPDATE_POLICY_NAME`, clear the selector by running the following command:

  ```
  d8 k patch moduleupdatepolicies.v1alpha1.deckhouse.io UPDATE_POLICY_NAME --type='json' -
  p='[{"op": "replace", "path":
  "/spec/moduleReleaseSelector/labelSelector/matchLabels", "value": {"": ""}}]'
  ```

- ModuleIsDeprecated CE S9
  ▾ **Module MODULE_NAME is deprecated.**

  The module `MODULE_NAME` is deprecated and will not receive new updates. The module may be removed in the future, so please disable it in a controlled manner.

- ModuleIsInMaintenanceMode CE S6
  ▾ **Module MODULE_NAME is running in maintenance mode.**

  Module `MODULE_NAME` is running in maintenance mode. In this mode, its state is not reconciled, which prevents configuration or hook changes from being applied automatically.

  To switch the module back to normal mode, edit the corresponding ModuleConfig resource with the following command:

  ```
  d8 k patch moduleconfig MODULE_NAME --type=json -p='[{"op": "remove", "path":
  "/spec/maintenance"}]'
  ```

- ModuleReleaseIsBlockedByRequirements CE S6
  ▾ **A new release for module MODULE_NAME has been blocked due to unmet requirements.**

  A new release for module `MODULE_NAME` has been blocked because it doesn’t meet the required conditions.

To check the requirements, run the following command:

```
d8 k  get mr NAME -o json | jq .spec.requirements
```

- ModuleReleaseIsOutdated CE S5
  ▼ Module MODULE_NAME is XXX or more minor versions behind the latest available release.

  Module `MODULE_NAME` is running XXX or more minor versions behind the latest available release. This module is significantly outdated and may have compatibility issues.

  To approve the module release update, run the following command:

  ```
  d8 k annotate mr NAME modules.deckhouse.io/approved="true"
  ```

- ModuleReleaseIsOutdated CE S7
  ▼ Module MODULE_NAME is 2 minor versions behind the latest available release.

  Module `MODULE_NAME` is running 2 minor versions behind the latest available release. Updating is recommended to maintain compatibility and access to latest features.

  To approve the module release update, run the following command:

  ```
  d8 k annotate mr NAME modules.deckhouse.io/approved="true"
  ```

- ModuleReleaseIsOutdated CE S9
  ▼ Module MODULE_NAME is 1 minor version behind the latest available release.

  Module `MODULE_NAME` is running 1 minor version behind the latest available release. Consider updating to maintain compatibility and security.

  To approve the module release update, run the following command:

  ```
  d8 k annotate mr NAME modules.deckhouse.io/approved="true"
  ```

- ModuleReleaseIsWaitingManualApproval CE S6
  ▼ A new release for module MODULE_NAME is awaiting manual approval.

  A new release for module `MODULE_NAME` is available but requires manual approval before it can be applied.

  To approve the module release, run the following command:

  ```
  d8 k annotate mr NAME modules.deckhouse.io/approved="true"
  ```

Module monitoring-kubernetes

- CPUStealHigh CE S4
  ▼ CPU steal on node NODE_NAME is too high.

  The CPU steal has been excessively high on node `NODE_NAME` over the past 30 minutes.

  Another component, such as a neighboring virtual machine, may be consuming the node’s resources. This may be the result of “overselling” the hypervisor, meaning the hypervisor is hosting more virtual machines than it can handle.

- DeadMansSwitch CE S4
  ▼ Alerting dead man's switch.

  This is a dead man’s switch meant to ensure that the entire Alerting pipeline is functional.

- DeploymentGenerationMismatch CE S4
  ▼ Deployment is outdated.

  The observed deployment generation doesn’t match the expected one for Deployment `NAMESPACE/DEPLOYMENT_NAME`.

- FdExhaustionClose CE S3
  ▼ File descriptors for JOB_NAME: INSTANCE_NAME are almost exhausted.

  The instance `JOB_NAME: INSTANCE_NAME` is expected to exhaust its available file/socket descriptors within the next hour.

- FdExhaustionClose CE S3
  ▼ **File descriptors for JOB_NAME: NAMESPACE/POD_NAME are almost exhausted.**

  The instance `JOB_NAME: NAMESPACE/POD_NAME` is expected to exhaust its available file/socket descriptors within the next hour.

- FdExhaustionClose CE S4
  ▼ **File descriptors for JOB_NAME: INSTANCE_NAME are exhausting soon.**

  The instance `JOB_NAME: INSTANCE_NAME` is expected to exhaust its available file/socket descriptors within the next 4 hours.

- FdExhaustionClose CE S4
  ▼ **File descriptors for JOB_NAME: NAMESPACE/POD_NAME are exhausting soon.**

  The instance `JOB_NAME: NAMESPACE/POD_NAME` is expected to exhaust its available file/socket descriptors within the next 4 hours.

- HelmReleasesHasResourcesWithDeprecatedVersions CE S5
  ▼ **At least one Helm release contains resources with deprecated apiVersion, which will be removed in Kubernetes version VERSION_NUMBER.**

  To list all affected resources, run the following Prometheus query:

  ```
  max by (helm_release_namespace, helm_release_name, helm_version,
  resource_namespace, resource_name, api_version, kind, k8s_version)
  (resource_versions_compatibility) == 1
  ```

  For more details on upgrading deprecated resources, refer to the Kubernetes deprecation guide available at `https://kubernetes.io/docs/reference/using-api/deprecation-guide/#vXXX`.

  Note that the check runs once per hour, so this alert should resolve within an hour after migrating deprecated resources.

- HelmReleasesHasResourcesWithUnsupportedVersions CE S4
  ▼ **At least one Helm release contains resources with unsupported apiVersion for Kubernetes version VERSION_NUMBER.**

  To list all affected resources, run the following Prometheus query:

  ```
  max by (helm_release_namespace, helm_release_name, helm_version,
  resource_namespace, resource_name, api_version, kind, k8s_version)
  (resource_versions_compatibility) == 2
  ```

  For more details on migrating deprecated resources, refer to the Kubernetes deprecation guide available at `https://kubernetes.io/docs/reference/using-api/deprecation-guide/#vXXX`.

  Note that the check runs once per hour, so this alert should resolve within an hour after migrating deprecated resources.

- K8SKubeletDown CE S3
  ▼ **Multiple kubelets couldn't be scraped.**

  Prometheus failed to scrape VALUE% of kubelets.

- K8SKubeletDown CE S4
  ▼ **Several kubelets couldn't be scraped.**

  Prometheus failed to scrape VALUE% of kubelets.

- K8SKubeletTooManyPods CE S7
  ▼ **The kubelet on node NODE_NAME is approaching the pod limit.**

  The kubelet on node NODE_NAME is running VALUE pods, which is close to the limit of XXX.

- K8SManyNodesNotReady CE S3
  ▼ **Too many nodes are not ready.**

  VALUE% of Kubernetes nodes are not ready.

- K8SNodeNotReady CE S3
  ▼ **The status of node NODE_NAME is NotReady.**

  The kubelet on node NODE_NAME has either failed to check in with the API server or has set itself

to `NotReady` for more than 10 minutes.

- KubeletImageFSBytesUsage CE S5

  No free space remaining on imagefs on node NODE_NAME at mountpoint MOUNTPOINT.

- KubeletImageFSBytesUsage CE S6
  ▾ Hard eviction of imagefs on node NODE_NAME at mountpoint MOUNTPOINT is in progress.

  Hard eviction of imagefs on node `NODE_NAME` at mountpoint `MOUNTPOINT` is currently in progress.

  Hard eviction threshold: XXX%

  Current usage: VALUE%

- KubeletImageFSBytesUsage CE S7
  ▾ Approaching hard eviction threshold of imagefs on node NODE_NAME at mountpoint MOUNTPOINT.

  The imagefs usage on node `NODE_NAME` at mountpoint `MOUNTPOINT` is nearing the hard eviction threshold.

  Hard eviction threshold: XXX%

  Current usage: VALUE%

- KubeletImageFSBytesUsage CE S9
  ▾ Soft eviction of imagefs on node NODE_NAME at mountpoint MOUNTPOINT is in progress.

  Soft eviction of imagefs on node `NODE_NAME` at mountpoint `MOUNTPOINT` is currently in progress.

  Soft eviction threshold: XXX%

  Current usage: VALUE%

- KubeletImageFSInodesUsage CE S5

  No free inodes remaining on imagefs on node NODE_NAME at mountpoint MOUNTPOINT.

- KubeletImageFSInodesUsage CE S6
  ▾ Hard eviction of imagefs on node NODE_NAME at mountpoint MOUNTPOINT is in progress.

  Hard eviction of imagefs on node `NODE_NAME` at mountpoint `MOUNTPOINT` is currently in progress.

  Hard eviction threshold: XXX%

  Current usage: VALUE%

- KubeletImageFSInodesUsage CE S7
  ▾ Approaching hard eviction threshold of imagefs on node NODE_NAME at mountpoint MOUNTPOINT.

  The imagefs usage on node `NODE_NAME` at mountpoint `MOUNTPOINT` is nearing the hard eviction threshold.

  Hard eviction threshold: XXX%

  Current usage: VALUE%

- KubeletImageFSInodesUsage CE S9
  ▾ Soft eviction of imagefs on node NODE_NAME at mountpoint MOUNTPOINT is in progress.

  Soft eviction of imagefs on node `NODE_NAME` at mountpoint `MOUNTPOINT` is currently in progress.

  Soft eviction threshold: XXX%

  Current usage: VALUE%

- KubeletNodeFSBytesUsage CE S5

  No free space remaining on nodefs on node NODE_NAME at mountpoint MOUNTPOINT.

- KubeletNodeFSBytesUsage CE S6
  ▾ Hard eviction of nodefs on node NODE_NAME at mountpoint MOUNTPOINT is in progress.

Hard eviction of nodefs on node `NODE_NAME` at mountpoint `MOUNTPOINT` is currently in progress.

Hard eviction threshold: XXX%

Current usage: VALUE%

- KubeletNodeFSBytesUsage CE S7
  ▼ Approaching hard eviction threshold of nodefs on node NODE_NAME at mountpoint MOUNTPOINT.

  The nodefs usage on node `NODE_NAME` at mountpoint `MOUNTPOINT` is nearing the hard eviction threshold.

  Hard eviction threshold: XXX%

  Current usage: VALUE%

- KubeletNodeFSBytesUsage CE S9
  ▼ Soft eviction of nodefs on node NODE_NAME at mountpoint MOUNTPOINT is in progress.

  Soft eviction of nodefs on node `NODE_NAME` at mountpoint `MOUNTPOINT` is currently in progress.

  Soft eviction threshold: XXX%

  Current usage: VALUE%

- KubeletNodeFSInodesUsage CE S5

  No free inodes remaining on nodefs on node NODE_NAME at mountpoint MOUNTPOINT.

- KubeletNodeFSInodesUsage CE S6
  ▼ Hard eviction of nodefs on node NODE_NAME at mountpoint MOUNTPOINT is in progress.

  Hard eviction of nodefs on node `NODE_NAME` at mountpoint `MOUNTPOINT` is currently in progress.

  Hard eviction threshold: XXX%

  Current usage: VALUE%

- KubeletNodeFSInodesUsage CE S7
  ▼ Approaching hard eviction threshold of nodefs on node NODE_NAME at mountpoint MOUNTPOINT.

  The nodefs usage on node `NODE_NAME` at mountpoint `MOUNTPOINT` is nearing the hard eviction threshold.

  Hard eviction threshold: XXX%

  Current usage: VALUE%

- KubeletNodeFSInodesUsage CE S9
  ▼ Soft eviction of nodefs on node NODE_NAME at mountpoint MOUNTPOINT is in progress.

  Soft eviction of nodefs on node `NODE_NAME` at mountpoint `MOUNTPOINT` is currently in progress.

  Soft eviction threshold: XXX%

  Current usage: VALUE%

- KubernetesDnsTargetDown CE S5
  ▼ Kube-dns or CoreDNS are not being monitored.

  Prometheus is unable to collect metrics from `kube-dns`, which makes its status unknown.

  Steps to troubleshoot:

  1. Check the deployment details:

     ```
     d8 k -n kube-system describe deployment -l k8s-app=kube-dns
     ```

  2. Check the pod details:

     ```
     d8 k -n kube-system describe pod -l k8s-app=kube-dns
     ```

- KubeStateMetricsDown CE S3
  ▼ **Kube-state-metrics isn't working in the cluster.**

  Deckhouse has detected that no metrics about cluster resources have been available for the past 5 minutes. As a result, most alerts and monitoring panels aren’t working.

  Steps to troubleshoot:

  1. Check the `kube-state-metrics` pods:

     ```
     d8 k -n d8-monitoring describe pod -l app=kube-state-metrics
     ```

  2. Check the deployment logs:

     ```
     d8 k -n d8-monitoring describe deploy kube-state-metrics
     ```

- LoadBalancerServiceWithoutExternalIP CE S4
  ▼ **A LoadBalancer has not been created.**

  One or more services of the `LoadBalancer` type have not received an external address.

  To list affected services, run the following command:

  ```
  d8 k get svc -Ao json | jq -r '.items[] | select(.spec.type == "LoadBalancer") |
  select(.status.loadBalancer.ingress[0].ip == null) | "namespace: \
  (.metadata.namespace), name: \(.metadata.name), ip: \
  (.status.loadBalancer.ingress[0].ip)"'
  ```

  Steps to troubleshoot:

  - Check the `cloud-controller-manager` logs in the `d8-cloud-provider-*` namespace.
  - If you are using a bare-metal cluster with the `metallb` module enabled, ensure the address pool has not been exhausted.

- NodeConntrackTableFull CE S3
  ▼ **The conntrack table on node NODE_NAME is full.**

  As a result, no new connections created or accepted on the node. Keeping the `conntrack` table at full capacity may lead to erratic software behavior.

  To identify the source of excessive `conntrack` entries, use Okmeter or Grafana dashboards.

- NodeConntrackTableFull CE S4
  ▼ **The conntrack table on node NODE_NAME is approaching the size limit.**

  The `conntrack` table on node `NODE_NAME` is currently at VALUE% of its maximum capacity.

  This is acceptable as long as the `conntrack` table remains 70-80% full. However, if it reaches full capacity, new connections may fail, causing network disruptions and erratic software behavior.

  To identify the source of excessive `conntrack` entries, use Okmeter or Grafana dashboards.

- NodeExporterDown CE S3
  ▼ **Prometheus couldn't scrape a node-exporter.**

  Prometheus has been unable to scrape a `node-exporter` for more than 10 minutes, or `node-exporters` have disappeared from service discovery.

- NodeFilesystemIsRO CE S4
  ▼ **Node file system is read-only.**

  The file system on the node has been switched to read-only mode.

  To investigate the cause, check the node logs.

- NodeSUnreclaimBytesUsageHigh CE S4
  ▼ **Node NODE_NAME has high kernel memory usage.**

  The node `NODE_NAME` has a potential kernel memory leak. One known issue could be causing this problem.

  Steps to troubleshoot:

  1. Check the `cgroupDriver` setting on node `NODE_NAME`:

```
cat /var/lib/kubelet/config.yaml | grep 'cgroupDriver: systemd'
```

2. If `cgroupDriver` is set to `systemd`, a reboot is required to switch back to the `cgroupfs` driver. In this case, drain and reboot the node.

For further details, refer to the issue.

- NodeSystemExporterDoesNotExistsForNode CE S4
  ▼ **Some system exporters aren't functioning correctly on node NODE_NAME.**

  To resolve the issue, follow these steps:

  1. Find the `node-exporter` Pod running on the affected node:

     ```
     d8 k -n d8-monitoring get pod -l app=node-exporter -o json | jq -r ".items[] |
     select(.spec.nodeName==\"NODE_NAME\") | .metadata.name"
     ```

  2. Describe the `node-exporter` Pod:

     ```
     d8 k -n d8-monitoring describe pod POD_NAME
     ```

  3. Verify that the kubelet is running on node `NODE_NAME`.

- NodeTCPMemoryExhaust CE S6
  ▼ **Node NODE_NAME has high TCP stack memory usage.**

  The TCP stack on node `NODE_NAME` is experiencing high memory pressure. This may be caused by applications with intensive TCP networking usage.

  Steps to troubleshoot:

  1. Identify applications consuming excessive TCP memory.
  2. Adjust TCP memory configuration if needed.
  3. Investigate network traffic sources.
- NodeUnschedulable CE S8
  ▼ **Node NODE_NAME is cordon-protected, preventing new pods from being scheduled.**

  The node `NODE_NAME` is in a cordon-protected state, meaning no new pods can be scheduled onto it.

  Someone may have executed one of the following commands on this node:

  - To cordon the node:

    ```
    d8 k cordon NODE_NAME
    ```

  - To drain the node (if draining has been running for more than 20 minutes):

    ```
    d8 k drain NODE_NAME
    ```

  The was likely caused by the scheduled maintenance of that node.

- PodOOMKilled CE S4
  ▼ **Pod was OOMKilled.**

  The container was killed by the Linux OOM Killer.

  - Namespace: NAMESPACE
  - Pod: POD_NAME
  - Container: XXX

  Check memory limits and actual memory usage.

- PodOOMKilled CE S4
  ▼ **Pod was OOMKilled.**

  The container was killed by the Linux OOM Killer.

  - Namespace: NAMESPACE
  - Pod: POD_NAME
  - Container: XXX

Check memory limits and actual memory usage.

- PodStatusIsIncorrect CE S6
  ▼ Incorrect state of Pod NAMESPACE/POD_NAME running on node NODE_NAME.

  The Pod `NAMESPACE/POD_NAME` running on node `NODE_NAME` is listed as `NotReady` while all the Pod's containers are `Ready`.

  This could have been caused by the known Kubernetes bug.

  Steps to troubleshoot:

  1. Identify all pods with this state:

     ```
     d8 k get pod -o json --all-namespaces | jq '.items[] | select(.status.phase == "Running") | select(.status.conditions[] | select(.type == "ContainersReady" and .status == "True")) | select(.status.conditions[] | select(.type == "Ready" and .status == "False")) | "\(.spec.nodeName)/\(.metadata.namespace)/\(.metadata.name)"'
     ```

  2. Identify all affected nodes:

     ```
     d8 k get pod -o json --all-namespaces | jq '.items[] | select(.status.phase == "Running") | select(.status.conditions[] | select(.type == "ContainersReady" and .status == "True")) | select(.status.conditions[] | select(.type == "Ready" and .status == "False")) | .spec.nodeName' -r | sort | uniq -c
     ```

  3. Restart kubelet on each node:

     ```
     systemctl restart kubelet
     ```

- StorageClassCloudManual CE S6
  ▼ Manually deployed StorageClass NAME found in the cluster.

  A StorageClass using a `cloud-provider` provisioner shouldn't be deployed manually. Such StorageClasses are managed by the `cloud-provider` module.

  Instead of the manual deployment, modify the `cloud-provider` module configuration as needed.

- StorageClassDefaultDuplicate CE S6
  ▼ Multiple default StorageClasses found in the cluster.

  Deckhouse has detected that more than one StorageClass in the cluster is annotated as default.

  This may have been caused by a manually deployed StorageClass that is overlapping with the default storage configuration provided by the `cloud-provider` module.

- UnsupportedContainerRuntimeVersion CE
  ▼ Unsupported version of CRI Containerd X.XX.XXX installed for Kubernetes version VERSION_NUMBER.

  Deckhouse has detected that the node NODE_NAME is running an unsupported version of CRI Containerd X.XX.XXX.

  Supported CRI versions for Kubernetes VERSION_NUMBER:

  - Containerd 1.4.
  - Containerd 1.5.
  - Containerd 1.6.
  - Containerd 1.7.

Module monitoring-kubernetes-control-plane

- K8SApiserverDown CE S3
  ▼ API servers can't be reached.

  No API servers are reachable, or they have all disappeared from service discovery.

- K8sCertificateExpiration CE S5
  ▼ Kubernetes has API clients with soon-to-expire certificates.

  Some clients are connecting to COMPONENT_NAME with certificates that will expire in less than a

Some clients are connecting to COMPONENT_NAME with certificates that will expire in less than a day on node `COMPONENT_NAME`.

To check control plane certificates, use kubeadm:

1. Install kubeadm using the following command:

   ```
   apt install kubeadm=1.24.*
   ```

2. Check certificates:

   ```
   kubeadm alpha certs check-expiration
   ```

To check kubelet certificates, do the following on each node:

1. Check kubelet configuration:

   ```
   ps aux \
     | grep "/usr/bin/kubelet" \
     | grep -o -e "--kubeconfig=\S*" \
     | cut -f2 -d"=" \
     | xargs cat
   ```

2. Locate the `client-certificate` or `client-certificate-data` field.
3. Check certificate expiration using OpenSSL.

Note that there are no tools to find other stale kubeconfig files. Consider enabling the `control-plane-manager` module for advanced debugging.

- K8sCertificateExpiration CE S6
  ▼ **Kubernetes has API clients with soon-to-expire certificates.**

  Some clients are connecting to COMPONENT_NAME with certificates that will expire in less than 7 days on node `NODE_NAME`.

  To check control plane certificates, use kubeadm:

  1. Install kubeadm using the following command:

     ```
     apt install kubeadm=1.24.*
     ```

  2. Check certificates:

     ```
     kubeadm alpha certs check-expiration
     ```

  To check kubelet certificates, do the following on each node:

  1. Check kubelet configuration:

     ```
     ps aux \
       | grep "/usr/bin/kubelet" \
       | grep -o -e "--kubeconfig=\S*" \
       | cut -f2 -d"=" \
       | xargs cat
     ```

  2. Locate the `client-certificate` or `client-certificate-data` field.
  3. Check certificate expiration using OpenSSL.

  Note that there are no tools to find other stale kubeconfig files. Consider enabling the `control-plane-manager` module for advanced debugging.

- K8SControllerManagerTargetDown CE S3
  ▼ **Controller manager is down.**

  There is no running kube-controller-manager. As a result, deployments and replication controllers are not progressing.

- K8SSchedulerTargetDown CE S3
  ▼ **Scheduler is down.**

  The Kubernetes scheduler is not running. As a result, new pods are not being assigned to nodes

The Kubernetes scheduler is not running. As a result, new pods are not being assigned to nodes.

- KubeEtcdHighFsyncDurations CE S7
  ▼ **Syncing (fsync) WAL files to disk is slow.**

  In the last 15 minutes, the 99th percentile of the fsync duration for WAL files exceeded 0.5 seconds: VALUE.

  Possible causes:

  - High disk latency where etcd data is stored.
  - High CPU usage on the node.
- KubeEtcdHighNumberOfLeaderChanges CE S5
  ▼ **The etcd cluster is re-electing the leader too frequently.**

  There have been VALUE leader re-elections for the etcd cluster member running on node `NODE_NAME` in the last 10 minutes.

  Possible causes:

  - High disk latency where etcd data is stored.
  - High CPU usage on the node.
  - Degradation of network connectivity between cluster members in the multi-master mode.
- KubeEtcdInsufficientMembers CE S4
  ▼ **Insufficient members in the etcd cluster.**

  The etcd cluster has too few members, increasing the risk of failure if another member becomes unavailable. To resolve this issue, check the status of etcd Pods: `bash d8 k -n kube-system get pod -l component=etcd`
- KubeEtcdNoLeader CE S4
  ▼ **The etcd cluster member running on node NODE_NAME has lost the leader.**

  To resolve this issue, check the status of the etcd Pods: `bash d8 k -n kube-system get pod -l component=etcd | grep NODE_NAME`
- KubeEtcdTargetAbsent CE S5
  ▼ **There is no etcd target in Prometheus.**

  Steps to troubleshoot:

  1. Check the status of the etcd Pods:

     ```
     d8 k -n kube-system get pod -l component=etcd
     ```

  2. Review Prometheus logs:

     ```
     d8 k -n d8-monitoring logs -l app.kubernetes.io/name=prometheus -c prometheus
     ```
- KubeEtcdTargetDown CE S5
  ▼ **Prometheus is unable to scrape etcd metrics.**

  Steps to troubleshoot:

  1. Check the status of the etcd Pods:

     ```
     d8 k -n kube-system get pod -l component=etcd
     ```

  2. Review Prometheus logs:

     ```
     d8 k -n d8-monitoring logs -l app.kubernetes.io/name=prometheus -c prometheus
     ```

Module monitoring-ping

- NodePingPacketLoss CE S4
  ▼ **Ping loss exceeds 5%.**

  ICMP packet loss to node `NODE_NAME` has exceeded 5%.

Module node-local-dns

- D8NodeLocalDNSCacheDenialEvictionsHigh BE S6

▾ node-local-dns denial cache is being frequently evicted.

The `denial` (NXDOMAIN) cache in node-local-dns is evicting entries frequently.

This might indicate that there are Services that are being recreated too often in cluster. So, denial cache size is not enough (`denial 65536` is currently configured).

Affected node: `NODE_NAME`

- D8NodeLocalDNSCacheSuccessEvictionsHigh BE S6
  ▾ node-local-dns success cache is being frequently evicted.

  The `success` cache in node-local-dns is evicting entries frequently (>100 in 5 minutes).

  This could mean there are too many different DNS records application have to resolve (to many Services?). So, node-local-dns cache capacity is too small (`success 120000` is currently configured), or traffic patterns changed.

  Affected node: `NODE_NAME`

- D8NodeLocalDNSForwardMaxConcurrentRejects BE S5
  ▾ The forward plugin is rejecting DNS queries due to max concurrency limit.

  More than 20 queries were rejected within the last 5 minutes on node `NODE_NAME` because the max concurrent query limit was reached.

- D8NodeLocalDNSKubeforwardRequestLatencyP95High BE S4
  ▾ High DNS request latency in kubeforward (p95 > 300ms under load).

  The 95th percentile DNS request latency for A-records through kubeforward exceeded 300 milliseconds over the last 5 minutes while handling more than 0.1 QPS.

  This may indicate slow upstreams, network issues, or DNS overload on this node.

  Affected node: `NODE_NAME`

  For more detailed diagnostics, inspect `Top 5 upstreams by slow kubeforward requests` in Grafana.

  If the alert fires only on one node, run the checks below from node `NODE_NAME` (for example, from a debug pod on this node). If the alert fires on many nodes, treat it as a potential systemic issue first.

  Get kubeforward upstream IPs from a master host:

  ```
  d8 k -n kube-system get po -l k8s-app=kube-dns -o jsonpath='{.items[*].status.podIP}'
  ```

  Then run latency checks on the affected node `NODE_NAME` (or from a debug pod on that node):

  ```
  UPSTREAMS="paste kube-dns pod IPs here"
  for ip in $UPSTREAMS; do
    echo "== upstream ${ip} =="
   for i in $(seq 1 20); do
      dig @"$ip" -p 5353 kubernetes.default.svc.cluster.local A +tries=1 +time=1 +stats | grep
  "Query time"
   done
  done
  ```

  To identify slow requests, enable module logs (`enableLogs`) and inspect slow query messages:

  ```
  pod=$(kubectl -n kube-system get po -l k8s-app=node-local-dns --field-selector
  spec.nodeName=NODE_NAME -o jsonpath='{.items[0].metadata.name}')
  d8 k -n kube-system logs "$pod" | grep '\[kubeforward\] slow query'
  ```

  Example log line: `[kubeforward] slow query A kubernetes.default.svc.cluster.local. took 425ms (rcode=NOERROR, upstream=10.96.0.10:53)`

  Interpretation:

  - If most upstreams are slow and many nodes are affected, suspect systemic network/CoreDNS overload.
  - If one upstream is slow or only one node is affected, suspect a localized issue.

- D8NodeLocalDNSUpstreamHealthcheckFailed BE S5
  ▼ **Too many upstream health check failures in forward plugin.**

  The health check system of the `forward` plugin failed more than 10 times within 5 minutes on node `NODE_NAME`. As a result, all upstreams were considered unhealthy.

Module node-manager

- CapsInstanceUnavailable CE S8
  ▼ **There are unavailable instances in the MACHINE_DEPLOYMENT_NAME MachineDeployment.**

  The MachineDeployment `MACHINE_DEPLOYMENT_NAME` has **VALUE** unavailable instances.

  Check the status of the instances in the cluster with the following command:

  ```
  d8 k get instance -l node.deckhouse.io/group=MACHINE_DEPLOYMENT_NAME
  ```

- ClusterHasOrphanedDisks CE S6
  ▼ **Cloud data discoverer found orphaned disks in the cloud.**

  The cloud data discoverer has found disks in the cloud that do not have a corresponding PersistentVolume in the cluster.

  You can safely delete these disks manually from your cloud provider:

  ID: ID, Name: NAME

- D8BashibleApiserverLocked CE S6
  ▼ **Bashible-apiserver has been locked for too long.**

  `Bashible-apiserver` has been locked for an extended period.

  To resolve the issue, check if the `bashible-apiserver` Pods are up-to-date and running:

  ```
  d8 k -n d8-cloud-instance-manager get pods -l app=bashible-apiserver
  ```

- D8CloudDataDiscovererCloudRequestError CE S6
  ▼ **Cloud data discoverer cannot get data from the cloud.**

  Refer to the cloud data discoverer’s logs for details:

  ```
  d8 k -n NAMESPACE logs deploy/cloud-data-discoverer
  ```

- D8CloudDataDiscovererSaveError CE S6
  ▼ **Cloud data discoverer cannot save data to a Kubernetes resource.**

  Refer to the cloud data discoverer’s logs for details:

  ```
  d8 k -n NAMESPACE logs deploy/cloud-data-discoverer
  ```

- D8ClusterAutoscalerManagerPodIsNotReady CE S8
  ▼ **The POD_NAME Pod is NOT Ready.**

  To check the Pod’s status, run the following command:

  ```
  d8 k -n NAMESPACE logs POD_NAME
  ```

- D8ClusterAutoscalerPodIsNotRunning CE S8
  ▼ **The cluster-autoscaler Pod is NOT Running.**

  The POD_NAME Pod is STATUS.

  To check the Pod’s status, run the following command:

  ```
  d8 k -n NAMESPACE get pods POD_NAME -o json | jq .status
  ```

- D8ClusterAutoscalerPodIsRestartingTooOften CE S9
  ▼ **Too many cluster-autoscaler restarts detected.**

  The cluster-autoscaler has restarted VALUE times in the past hour.

Frequent restarts may indicate a problem. The cluster-autoscaler is expected to run continuously without interruption.

Check the logs for details:

```
d8 k -n d8-cloud-instance-manager logs -f -l app=cluster-autoscaler -c cluster-autoscaler
```

- D8ClusterAutoscalerTargetAbsent CE S8
  ▼ **Cluster-autoscaler target is missing in Prometheus.**

  The cluster-autoscaler automatically scales nodes in the cluster. If it’s unavailable, it will be impossible to add new nodes when there’s not enough resources for scheduling Pods. It may also lead to unnecessary cloud costs due to unused but still provisioned cloud instances.

  To resolve the issue, follow these steps:

  1. Check availability and status of cluster-autoscaler Pods:

     ```
     d8 k -n d8-cloud-instance-manager get pods -l app=cluster-autoscaler
     ```

  2. Verify that the cluster-autoscaler Deployment exists:

     ```
     d8 k -n d8-cloud-instance-manager get deploy cluster-autoscaler
     ```

  3. Check the Deployment’s status:

     ```
     d8 k -n d8-cloud-instance-manager describe deploy cluster-autoscaler
     ```

- D8ClusterAutoscalerTargetDown CE S8

  Prometheus is unable to scrape cluster-autoscaler’s metrics.

- D8ClusterAutoscalerTooManyErrors CE S8
  ▼ **Cluster-autoscaler is issuing too many errors.**

  The cluster-autoscaler encountered multiple errors from the cloud provider when attempting to scale the cluster.

  Check the logs for details:

  ```
  d8 k -n d8-cloud-instance-manager logs -f -l app=cluster-autoscaler -c cluster-autoscaler
  ```

- D8MachineControllerManagerPodIsNotReady CE S8

  The POD_NAME Pod is NOT Ready.

- D8MachineControllerManagerPodIsNotRunning CE S8
  ▼ **The machine-controller-manager Pod is NOT Running.**

  The POD_NAME Pod is STATUS.

  To check the Pod’s status, run the following command:

  ```
  d8 k -n NAMESPACE get pods POD_NAME -o json | jq .status
  ```

- D8MachineControllerManagerPodIsRestartingTooOften CE S9
  ▼ **Too many machine-controller-manager restarts detected.**

  The `machine-controller-manager` has restarted VALUE times in the past hour.

  Frequent restarts may indicate a problem. The `machine-controller-manager` is expected to run continuously without interruption.

  Check the logs for details:

  ```
  d8 k -n d8-cloud-instance-manager logs -f -l app=machine-controller-manager -c controller
  ```

- D8MachineControllerManagerTargetAbsent CE S8
  ▼ **Machine-controller-manager target is missing in Prometheus.**

  `Machine-controller-manager` controls ephemeral nodes in the cluster. If it becomes

unavailable, it will be impossible to create or delete nodes in the cluster.

To resolve the issue, follow these steps:

1. Check availability and status of `machine-controller-manager` Pods:

   ```
   d8 k -n d8-cloud-instance-manager get pods -l app=machine-controller-manager
   ```

2. Verify availability of the `machine-controller-manager` Deployment:

   ```
   d8 k -n d8-cloud-instance-manager get deployment machine-controller-manager
   ```

3. Check the Deployment’s status:

   ```
   d8 k -n d8-cloud-instance-manager describe deployment machine-controller-manager
   ```

- D8MachineControllerManagerTargetDown CE S8

  Prometheus is unable to scrape the machine-controller-manager’s metrics.

- D8NeedMigrateStateToOpenTofu CE S4
  ▼ Terraform state must be migrated to OpenTofu.

  This likely means the automatic migration to OpenTofu was canceled due to destructive changes detected in the cluster state.

  Verify the following:

  - Current Terraform version:

    ```
    d8 k -n d8-system exec -it deployments/terraform-state-exporter -c exporter -- terraform version
    ```

  - Version in the Terraform state:

    ```
    d8 k -n d8-system get secret d8-cluster-terraform-state -o json | jq -r '.data["cluster-tf-state.json"] | @base64d | fromjson | .terraform_version'
    ```

  - Check for destructive changes:

    ```
    d8 k exec -it deploy/terraform-state-exporter -n d8-system -- dhctl terraform check
    ```

  To resolve the issue and migrate to OpenTofu manually, follow these steps:

  1. Using the `install` container with the previous major Deckhouse release (for example, `1.69.X`, while the cluster is now on `1.70.X`), run:

     ```
     dhctl converge
     ```

     This should resolve all destructive changes in the cluster.

  2. After that, the `terraform-auto-converger` Pod should complete the migration automatically.

  If the automatic migration doesn’t happen, use the `install` container with the current Deckhouse release and run the following command to enforce migration of the Terraform state to OpenTofu:

  ```
  dhctl converge-migration
  ```

  If this alert appears in a cluster created by Deckhouse Commander, it means the cluster has been updated to a new Deckhouse version without approving state changes (for example, destructive changes).

  To resolve this, follow these steps:

  1. In Deckhouse Commander, click **Delete manually** to temporarily remove the cluster from control.

2. In the cluster, remove the `commander-agent` module using the following command:

   ```
   d8 k delete mc commander-agent
   ```

3. Using the `install` container with the previous major Deckhouse release (for example, `1.69.X`, while the cluster is now on `1.70.X`), run the following command to resolve all destructive changes in the cluster:

   ```
   dhctl converge
   ```

4. In Deckhouse Commander, click **Attach** to reattach the cluster.

- D8NodeCgroupV2NotSupported CE S8
  ▼ **Node NODE_NAME is using deprecated cgroup v1 – migration to v2 required.**

  Node NODE_NAME in NodeGroup NODE_GROUP_NAME does not support cgroup v2. Starting from Kubernetes 1.35, cgroup v1 is deprecated. To migrate to cgroup v2, please upgrade the kernel to version 5.8 or newer. For more information, please refer to the documentation.

- D8NodeContainerdV2NotSupported CE S8
  ▼ **Node NODE_NAME is using deprecated containerd v1 – migration to v2 required.**

  Node NODE_NAME in NodeGroup NODE_GROUP_NAME does not support containerd v2. Starting from Kubernetes 1.37, containerd v1.y will no longer be supported. Please check the requirements for containerd v2 and schedule your migration plan.

  The exact reason is listed in the `node.deckhouse.io/containerd-v2-err` annotation on the node.

  If the reason is `erofs` and the node runs Ubuntu in AWS, the module is shipped in a separate `linux-modules-extra` package. DKP installs it automatically, but for some kernels, mostly outdated ones, the package is no longer published in the APT repository. In that case, the installation is skipped and a warning is written to the node logs. The problem has been confirmed for the `5.15.0-1028-aws`, `6.8.0-1024-aws`, and `6.8.0-1029-aws` kernels. Switch the node group to a newer AMI or kernel version that includes `erofs`.

- D8NodeGroupIsNotUpdating CE S8
  ▼ **NodeGroup NODE_GROUP_NAME is not handling the update correctly.**

  There is a new update available for Nodes in the `NODE_GROUP_NAME` NodeGroup. Although Nodes have detected the update, none of them have received approval to start the update process.

  Most likely, there is a problem with the `update_approval` hook of the `node-manager` module.

- D8NodeIsNotUpdating CE S7
  ▼ **Node NODE_NAME cannot complete the update.**

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` detected a new update, requested and received approval, started the update, and encountered a step that could cause downtime. The update manager (the `update_approval` hook of the `node-group` module) granted downtime approval, but no success message was received, which indicates that the update has not completed.

  To investigate the details, view Bashible logs on the Node:

  ```
  journalctl -fu bashible
  ```

- D8NodeIsNotUpdating CE S8
  ▼ **Node NODE_NAME cannot complete the update.**

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` has detected a new update, requested and received approval, but failed to complete the update.

  To investigate the details, view Bashible logs on the Node:

  ```
  journalctl -fu bashible
  ```

- D8NodeIsNotUpdating CE S9
  ▼ **Node NODE_NAME is not updating.**

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` has a pending update but it’s neither receiving it nor attempting to.

  Most likely, Bashible is not handling the update correctly. It should annotate the Node with

`update.node.deckhouse.io/waiting-for-approval` before the update can proceed.

Options to investigate the details:

- Check the expected configuration checksum for the NodeGroup:

```
d8 k -n d8-cloud-instance-manager get secret configuration-checksums -o jsonpath=
{.data.NODE_GROUP_NAME} | base64 -d
```

- Check the current configuration checksum on the Node:

```
d8 k get node NODE_NAME -o
jsonpath='{.metadata.annotations.node\.deckhouse\.io/configuration-
checksum}'
```

- View Bashible logs on the Node:

```
journalctl -fu bashible
```

- D8NodeIsUnmanaged CE S9
  ▼ The NODE_NAME Node is not managed by the node-manager module.

  The NODE_NAME Node is not managed by the `node-manager` module.

  To resolve this issue, follow the instructions how to clean up the node and add it to the cluster.

- D8NodeUpdateStuckWaitingForDisruptionApproval CE S8
  ▼ Node NODE_NAME cannot obtain disruption approval.

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` detected a new update, requested and received initial approval, and started the update. However, it reached a stage that could cause downtime and was unable to obtain disruption approval. A disruption approval is normally issued automatically by the `update_approval` hook of the `node-manager` module.

  To resolve this issue, investigate why the approval could not be granted to proceed with the update.

- D8ProblematicNodeGroupConfiguration CE S8
  ▼ Node NODE_NAME cannot begin the update.

  There is a new update available for Nodes in the `NODE_GROUP_NAME` NodeGroup. However, Node `NODE_NAME` cannot begin the update.

  The Node is missing the `node.deckhouse.io/configuration-checksum` annotation, which may indicate that its bootstrap process did not complete correctly.

  Troubleshooting options:

  - Check the `cloud-init` log (`/var/log/cloud-init-output.log`) on the node.
  - Check the NodeGroupConfiguration resource associated with the `NODE_GROUP_NAME` NodeGroup for potential issues.
- EarlyOOMPodIsNotReady CE S8
  ▼ Pod POD_NAME has detected an unavailable PSI subsystem.

  The Pod POD_NAME detected that the Pressure Stall Information (PSI) subsystem is unavailable.

  For details, check the logs:

```
d8 k -n d8-cloud-instance-manager logs POD_NAME
```

  Troubleshooting options:

  - Upgrade the Linux kernel to version 4.20 or higher.
  - Enable the Pressure Stall Information.
  - Disable early OOM.
- NodeGroupHasStaticInternalNetworkCIDRsField CE S9
  ▼ NodeGroup NAME uses deprecated field spec.static.internalNetworkCIDRs.

  The `spec.static.internalNetworkCIDRs` field is deprecated and has been moved to the static cluster configuration.

  To resolve this issue, delete the field from NodeGroup NAME. This is safe, as the setting has

already been migrated automatically.

- NodeGroupMasterTaintIsAbsent CE S4
  ▼ The master NodeGroup is missing the required control-plane taint.

  The `master` NodeGroup doesn't have the `node-role.kubernetes.io/control-plane: NoSchedule` taint. This may indicate a misconfiguration where control-plane nodes can run non-control-plane Pods.

  To resolve the issue, add the following to the `master` NodeGroup spec:

  ```
  nodeTemplate:
    taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/control-plane
  ```

  Note that the taint `key: node-role.kubernetes.io/master` is deprecated and has no effect starting from Kubernetes 1.24.

- NodeGroupReplicasUnavailable CE S7
  ▼ NodeGroup NODE_GROUP_NAME has no available instances.

  This probably means that `machine-controller-manager` is unable to create Machines using the cloud provider module.

  Possible causes:

  1. Cloud provider resource limits.
  2. No access to the cloud provider API.
  3. Misconfiguration of the cloud provider or instance class.
  4. Problems with bootstrapping the Machine.

  Recommended actions:

  1. Check the status of the NodeGroup:

     ```
     d8 k get ng NODE_GROUP_NAME -o yaml
     ```

     Look for errors in the `.status.lastMachineFailures` field.

  2. If no Instances stay in the Pending state for more than a couple of minutes, it likely means that Instances are being continuously created and deleted due to an error:

     ```
     d8 k get instances
     ```

  3. If logs don't show errors, and an Instance continues to be Pending, check its bootstrap status:

     ```
     d8 k get instances INSTANCE_NAME -o yaml | grep 'bootstrapStatus' -B0 -A2
     ```

  4. If the output looks like the example below, use the `logsEndpoint` (or the command in `description`) to examine bootstrap logs:

     ```
     bootstrapStatus:
       description: Use 'curl -N http://192.168.199.158:8000' to get bootstrap logs.
       logsEndpoint: http://192.168.199.158:8000
     ```

  5. If there's no bootstrap log endpoint, `cloudInit` may not be working correctly. This could indicate a misconfigured instance class in the cloud provider.

- NodeGroupReplicasUnavailable CE S8
  ▼ Too many unavailable instances in the NODE_GROUP_NAME NodeGroup.

  The number of simultaneously unavailable instances in the NODE_GROUP_NAME NodeGroup exceeds the allowed threshold. This may indicate that the autoscaler has provisioned too many nodes at once. Check the state of the Machine in the cluster. This probably means that `machine-controller-manager` is unable to create Machines using the cloud provider module.

  Possible causes:

  1. Cloud provider resource limits.

2. No access to the cloud provider API.
3. Misconfiguration of the cloud provider or instance class.
4. Problems with bootstrapping the Machine.

Recommended actions:

1. Check the status of the NodeGroup:

   ```
   d8 k get ng NODE_GROUP_NAME -o yaml
   ```

   Look for errors in the `.status.lastMachineFailures` field.

2. If no Instances stay in the Pending state for more than a couple of minutes, it likely means that Instances are being continuously created and deleted due to an error:

   ```
   d8 k get instances
   ```

3. If logs don't show errors, and an Instance continues to be Pending, check its bootstrap status:

   ```
   d8 k get instances INSTANCE_NAME -o yaml | grep 'bootstrapStatus' -B0 -A2
   ```

4. If the output looks like the example below, use the `logsEndpoint` (or the command in `description`) to examine bootstrap logs:

   ```
   bootstrapStatus:
     description: Use 'curl -N http://192.168.199.158:8000' to get bootstrap logs.
     logsEndpoint: http://192.168.199.158:8000
   ```

5. If there's no bootstrap log endpoint, `cloudInit` may not be working correctly. This could indicate a misconfigured instance class in the cloud provider.

- NodeGroupReplicasUnavailable CE S8
  ▼ **NodeGroup NODE_GROUP_NAME has unavailable instances.**

  There are VALUE unavailable instances in the NodeGroup NODE_GROUP_NAME. This probably means that `machine-controller-manager` is unable to create Machines using the cloud provider module.

  Possible causes:

  1. Cloud provider resource limits.
  2. No access to the cloud provider API.
  3. Misconfiguration of the cloud provider or instance class.
  4. Problems with bootstrapping the Machine.

  Recommended actions:

  1. Check the status of the NodeGroup:

     ```
     d8 k get ng NODE_GROUP_NAME -o yaml
     ```

     Look for errors in the `.status.lastMachineFailures` field.

  2. If no Instances stay in the Pending state for more than a couple of minutes, it likely means that Instances are being continuously created and deleted due to an error:

     ```
     d8 k get instances
     ```

  3. If logs don't show errors, and an Instance continues to be Pending, check its bootstrap status:

     ```
     d8 k get instances INSTANCE_NAME -o yaml | grep 'bootstrapStatus' -B0 -A2
     ```

  4. If the output looks like the example below, use the `logsEndpoint` (or the command in `description`) to examine bootstrap logs:

```
bootstrapStatus:
  description: Use 'curl -N http://192.168.199.158:8000' to get bootstrap logs.
  logsEndpoint: http://192.168.199.158:8000
```

5. If there's no bootstrap log endpoint, `cloudInit` may not be working correctly. This could indicate a misconfigured instance class in the cloud provider.

- NodeRequiresDisruptionApprovalForUpdate CE S8
  ▾ **Node NODE_NAME requires disruption approval to proceed with the update.**

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` has detected a new update, received initial approval, and started the update. However, it encountered a stage that may cause downtime and requires manual disruption approval, because the NodeGroup is configured with `disruptions.approvalMode: Manual`.

  To resolve the issue, ensure the Node is ready for unsafe updates (drained) and grant disruption approval by annotating the Node with `update.node.deckhouse.io/disruption-approved=`.

  Caution:

  - Nodes in manual mode aren't drained automatically.
  - Do not drain master nodes.

  1. To drain the Node and grant the update approval, run the following command:

     ```
     d8 k drain NODE_NAME --delete-emptydir-data --ignore-daemonsets --force=true &&
       d8 k annotate node NODE_NAME update.node.deckhouse.io/disruption-approved=
     ```

  2. Uncordon the Node once the update is complete and the annotation `update.node.deckhouse.io/approved` is removed:

     ```
     while d8 k get node NODE_NAME -o json | jq -e '.metadata.annotations |
     has("update.node.deckhouse.io/approved")' > /dev/null; do sleep 1; done
     d8 k uncordon NODE_NAME
     ```

  If the NodeGroup has multiple Nodes, repeat this process for each one, since only one Node is updated at a time. Consider temporarily switching to automatic disruption approval (`disruptions.approvalMode: Automatic`).

- NodeStuckInDraining CE S6
  ▾ **Node NODE_NAME is stuck in draining.**

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` is stuck in the draining process.

  To get more details, run the following:

  ```
  d8 k -n default get event --field-selector involvedObject.name=NODE_NAME,reason=DrainFailed --
  sort-by='.metadata.creationTimestamp'
  ```

  Error message: MESSAGE_CONTENTS

- NodeStuckInDrainingForDisruptionDuringUpdate CE S6
  ▾ **Node NODE_NAME is stuck in draining.**

  Node `NODE_NAME` in NodeGroup `NODE_GROUP_NAME` has detected a new update, requested and received approval, started the update, and reached a step that could cause downtime. It is currently stuck in the draining process while waiting for automatic disruption approval.

  To get more details, run the following:

  ```
  d8 k -n default get event --field-selector involvedObject.name=NODE_NAME,reason=ScaleDown --
  sort-by='.metadata.creationTimestamp'
  ```

- StandbyHolderDeploymentReplicasUnavailable CE S5
  ▾ **No available replicas in Deployment standby-holder-NODE_GROUP_NAME.**

  Deckhouse has detected that there are no available replicas in the `standby-holder-NODE_GROUP_NAME` Deployment in the `d8-cloud-instance-manager` namespace.

  Check the Deployment and associated Pods for issues.

- UnmetCloudConditions CE S6
  ▼ **Deckhouse update is unavailable due to unmet cloud provider conditions.**

  Deckhouse has detected that some cloud provider–specific conditions have not been met. Until these issues are resolved, updating to the new Deckhouse release is not possible.

  Troubleshooting details:

  - Name: NAME
  - Message: MESSAGE_CONTENTS

Module okmeter

- D8OkmeterAgentPodIsNotReady CE S6

  Okmeter agent is not Ready.

Module openvpn

- OpenVPNClientCertificateExpired CE S4
  ▼ **OpenVPN client certificate expired for XXX**

  The OpenVPN client certificate for **XXX** has expired.

  Check the certificate information and make sure it actually expires:

  ```
  d8 k -n d8-openvpn get secrets -l name="XXX"  -o jsonpath='{.items[0].data.tls\.crt}' | base64 -d | openssl x509 -text -noout
  ```

  Renew or delete the expired certificate. Check the documentation for details.

- OpenVPNClientCertificateExpired CE S5
  ▼ **OpenVPN client certificate expires for XXX in less than 7 days.**

  The OpenVPN client certificate for **XXX** is set to expire in VALUE days.

  Renewal of certificate required. Check the documentation for details.

  View certificate details to display the exact expiration date:

  ```
  d8 k -n d8-openvpn get secrets -l name=XXX  -o jsonpath='{.items[0].data.tls\.crt}' | base64 -d | openssl x509 -text -noout
  ```

- OpenVPNClientCertificateExpired CE S6
  ▼ **OpenVPN client certificate expires for XXX in less than 30 days.**

  The OpenVPN client certificate for **XXX** is set to expire in VALUE days.

  Renewal of certificate required. Check the documentation for details.

  View certificate details to display the exact expiration date:

  ```
  d8 k -n d8-openvpn get secrets -l name=XXX  -o jsonpath='{.items[0].data.tls\.crt}' | base64 -d | openssl x509 -text -noout
  ```

- OpenVPNServerCACertificateExpired CE S4
  ▼ **OpenVPN CA certificate has expired.**

  OpenVPN CA certificate renews automatically 1 days before expiration. Automatic rotation did not work. Check the module status or perform manual rotation.

  View certificate details to display the exact expiration date:

  ```
  d8 k -n d8-openvpn get secrets openvpn-pki-ca  -o jsonpath='{.data.tls\.crt}' | base64 -d | openssl x509 -text -noout
  ```

  The hook should rotate CA and server certificates before expiry.

- OpenVPNServerCACertificateExpiresTomorrow CE S4
  ▼ **OpenVPN CA certificate will expire tomorrow**

  OpenVPN CA certificate renews automatically 1 days before expiration. Automatic rotation did not

work. Check the module status or perform manual rotation.

View certificate details to display the exact expiration date:

```
d8 k -n d8-openvpn get secrets openvpn-pki-ca  -o jsonpath='{.data.tls\.crt}' | base64 -d
| openssl x509 -text -noout
```

The hook should rotate CA and server certificates before expiry.

- OpenVPNServerCACertificateExpiringInAWeek CE S5
  ▼ **OpenVPN CA certificate expires in less than 7 days.**

  The OpenVPN CA certificate is set to expire in VALUE days.

  Renew the CA certificate if necessary. After renewing the certificate, **you will need to reissue all client and server certificates**. Check the documentation for details.

  View certificate details to display the exact expiration date:

  ```
  d8 k -n d8-openvpn get secrets -l name=server  -o jsonpath='{.items[0].data.tls\.crt}' |
  base64 -d | openssl x509 -text -noout
  ```

- OpenVPNServerCACertificateExpiringSoon CE S5
  ▼ **OpenVPN CA certificate expires in less than 30 days.**

  The OpenVPN CA certificate is set to expire in VALUE days.

  Renew the CA certificate if necessary. After renewing the certificate, **you will need to reissue all client and server certificates**. Check the documentation for details.

  View certificate details to display the exact expiration date:

  ```
  d8 k -n d8-openvpn get secrets -l name=server  -o jsonpath='{.items[0].data.tls\.crt}' |
  base64 -d | openssl x509 -text -noout
  ```

- OpenVPNServerCertificateExpired CE S4
  ▼ **OpenVPN server certificate has expired.**

  The OpenVPN server certificate has expired.

  OpenVPN server certificate renews automatically 1 days before expiration. Automatic rotation did not work. Check the module status or perform manual rotation.

  Check the certificate information and make sure it actually expires.

  ```
  d8 k -n d8-openvpn get secrets -l name=server  -o jsonpath='{.items[0].data.tls\.crt}' |
  base64 -d | openssl x509 -text -noout
  ```

- OpenVPNServerCertificateExpiresTomorrow CE S4
  ▼ **OpenVPN server certificate will expire tomorrow.**

  The OpenVPN server certificate is set to expire in less than 1 day.

  OpenVPN server certificate renews automatically 1 days before expiration. Automatic rotation did not work. Check the module status or perform manual rotation.

  Check the certificate information and make sure it actually expires:

  ```
  d8 k -n d8-openvpn get secrets -l name=server  -o jsonpath='{.items[0].data.tls\.crt}' |
  base64 -d | openssl x509 -text -noout
  ```

Module operator-prometheus

- D8PrometheusOperatorPodIsNotReady CE S7
  ▼ **The prometheus-operator Pod is NOT Ready.**

  As a result, new `Prometheus`, `PrometheusRules`, and `ServiceMonitor` configurations cannot be applied in the cluster. However, all existing and configured components will continue to operate normally. This problem will not affect alerting or monitoring in the short term (for a few days).

  Troubleshooting steps:

1. Analyze the Deployment details:

```
d8 k -n d8-operator-prometheus describe deployment prometheus-operator
```

2. Examine the Pod’s to determine why it is not running:

```
d8 k -n d8-operator-prometheus describe pod -l app=prometheus-operator
```

- D8PrometheusOperatorPodIsNotRunning CE S7
  ▼ **The prometheus-operator Pod is NOT Running.**

  As a result, new `Prometheus`, `PrometheusRules`, and `ServiceMonitor` configurations cannot be applied in the cluster. However, all existing and configured components will continue to operate normally. This problem will not affect alerting or monitoring in the short term (for a few days).

  Troubleshooting steps:

  1. Analyze the Deployment details:

```
d8 k -n d8-operator-prometheus describe deployment prometheus-operator
```

  2. Examine the Pod’s to determine why it is not running:

```
d8 k -n d8-operator-prometheus describe pod -l app=prometheus-operator
```

- D8PrometheusOperatorTargetAbsent CE S7
  ▼ **Prometheus-operator target is missing in Prometheus.**

  As a result, new `Prometheus`, `PrometheusRules`, and `ServiceMonitor` configurations cannot be applied in the cluster. However, all existing and configured components will continue to operate normally. This problem will not affect alerting or monitoring in the short term (for a few days).

  To resolve the issue, analyze the Deployment details:

```
d8 k -n d8-operator-prometheus describe deployment prometheus-operator
```

- D8PrometheusOperatorTargetDown CE S8
  ▼ **Prometheus is unable to scrape prometheus-operator metrics.**

  The `prometheus-operator` Pod is unavailable.

  As a result, new `Prometheus`, `PrometheusRules`, and `ServiceMonitor` configurations cannot be applied in the cluster. However, all existing and configured components will continue to operate normally. This problem will not affect alerting or monitoring in the short term (for a few days).

  Troubleshooting steps:

  1. Analyze the Deployment details:

```
d8 k -n d8-operator-prometheus describe deployment prometheus-operator
```

  2. Examine the Pod’s to determine why it is not running:

```
d8 k -n d8-operator-prometheus describe pod -l app=prometheus-operator
```

Module prometheus

- D8GrafanaDeploymentReplicasUnavailable CE S6
  ▼ **One or more Grafana Pods are NOT Running.**

  The number of Grafana replicas is lower than the specified number.

  The Deployment is in the `MinimumReplicasUnavailable` state.

  Troubleshooting options:

  - To check the Deployment’s status:

```
d8 k -n d8-monitoring get deployment grafana-v10 -o json | jq .status
```

  - To check a Pod's status:

```
d8 k -n d8-monitoring get pods -l app=grafana-v10 -o json | jq '.items[] |
{(.metadata.name):.status}'
```

- D8GrafanaDeprecatedCustomDashboardDefinition CE S9
  ▼ **Deprecated ConfigMap for Grafana dashboards detected.**

  The ConfigMap `grafana-dashboard-definitions-custom` has been found in the `d8-monitoring` namespace. This indicates that a deprecated method for registering custom dashboards in Grafana is used.

  This method is no longer supported.

  Migrate to using the custom GrafanaDashboardDefinition resource instead.

- D8GrafanaPodIsNotReady CE S6

  The Grafana Pod is NOT Ready.

- D8GrafanaPodIsRestartingTooOften CE S9
  ▼ **Excessive Grafana restarts detected.**

  Grafana has restarted VALUE times in the last hour.

  Frequent restarts indicate a problem. Grafana is expected to run continuously without interruption.

  To investigate the issue, check the logs:

```
d8 k -n d8-monitoring logs -f -l app=grafana-v10 -c grafana
```

- D8GrafanaTargetAbsent CE S6
  ▼ **Grafana target is missing in Prometheus.**

  Grafana visualizes metrics collected by Prometheus. Grafana is critical for some tasks, such as monitoring the state of applications and the cluster as a whole. Additionally, Grafana's unavailability can negatively impact users who actively use it in their work.

  The recommended course of action:

  1. Check the availability and status of Grafana Pods:

```
d8 k -n d8-monitoring get pods -l app=grafana-v10
```

  2. Check the availability of the Grafana Deployment:

```
d8 k -n d8-monitoring get deployment grafana-v10
```

  3. Examine the status of the Grafana Deployment:

```
d8 k -n d8-monitoring describe deployment grafana-v10
```

- D8GrafanaTargetDown CE S6

  Prometheus is unable to scrape Grafana metrics.

- D8PrometheusConfigReloadFailed CE S5
  ▼ **Prometheus configuration reload failed.**

  The Prometheus configuration reload has failed in the `d8-monitoring` namespace.

  This usually happens when:

  - Invalid scrape configuration (e.g., from ServiceMonitor/PodMonitor resources).
  - Invalid RemoteWrite configuration.

  Prometheus will continue running with the old configuration, but new rules, targets, or remote write endpoints will not be active. Troubleshooting steps:

1. Check the Prometheus container logs for configuration errors:

   ```
   d8 k -n d8-monitoring logs prometheus-main-0 -c prometheus
   ```

2. If using remote write, verify the remote write configuration.

Note: In HA configurations with multiple Prometheus replicas, check prometheus-main-1 logs as well.

- D8PrometheusLongtermFederationTargetDown CE S5
  ▼ **Prometheus-longterm cannot scrape Prometheus.**

  The `prometheus-longterm` instance is unable to scrape the `/federate` endpoint from Prometheus.

  Troubleshooting options:

  - Check the `prometheus-longterm` logs.
  - Open the corresponding web UI to check scrape errors.
- D8PrometheusLongtermTargetAbsent CE S7
  ▼ **Prometheus-longterm target is missing in Prometheus.**

  The `prometheus-longterm` component is used to display historical monitoring data and is not crucial. However, its extended downtime may prevent access to statistics.

  This issue is often caused by problems with disk availability. For example, if the disk cannot be mounted to a Node.

  Troubleshooting steps:

  1. Check the StatefulSet status:

     ```
     d8 k -n d8-monitoring describe statefulset prometheus-longterm
     ```

  2. Inspect the PersistentVolumeClaim (if used):

     ```
     d8 k -n d8-monitoring describe pvc prometheus-longterm-db-prometheus-longterm-0
     ```

  3. Inspect the Pod’s state:

     ```
     d8 k -n d8-monitoring describe pod prometheus-longterm-0
     ```

- D8TricksterTargetAbsent CE S5
  ▼ **Trickster target is missing in Prometheus.**

  The following modules use the Trickster component:

  - `prometheus-metrics-adapter`: Its unavailability means horizontal pod autoscaling (HPA) is not working, and you cannot view resource consumption using `d8 k`.
  - `vertical-pod-autoscaler`: Short-term unavailability for this module is tolerable, as VPA looks at the consumption history for 8 days.
  - `grafana`: All dashboards use Trickster by default to cache Prometheus queries. You can retrieve data directly from Prometheus (bypassing the Trickster). However, this may lead to high memory usage by Prometheus and cause unavailability.

  Troubleshooting steps:

  1. Inspect the Deployment’s stats:

     ```
     d8 k -n d8-monitoring describe deployment trickster
     ```

  2. Inspect the Pod’s stats:

     ```
     d8 k -n d8-monitoring describe pod -l app=trickster
     ```

  3. Trickster often becomes unavailable due to Prometheus issues, since its `readinessProbe` depends on Prometheus being accessible.

     Make sure Prometheus is running:

```
d8 k -n d8-monitoring describe pod -l app.kubernetes.io/name=prometheus,prometheus=main
```

- D8TricksterTargetAbsent CE S5
  ▼ **Trickster target is missing in Prometheus.**

  The following modules use the Trickster component:

  - `prometheus-metrics-adapter`: Its unavailability means horizontal pod autoscaling (HPA) is not working, and you cannot view resource consumption using `d8 k`.
  - `vertical-pod-autoscaler`: Short-term unavailability for this module is tolerable, as VPA looks at the consumption history for 8 days.
  - `grafana`: All dashboards use Trickster by default to cache Prometheus queries. You can retrieve data directly from Prometheus (bypassing the Trickster). However, this may lead to high memory usage by Prometheus and cause unavailability.

  Troubleshooting steps:

  1. Inspect the Deployment's stats:

     ```
     d8 k -n d8-monitoring describe deployment trickster
     ```

  2. Inspect the Pod's stats:

     ```
     d8 k -n d8-monitoring describe pod -l app=trickster
     ```

  3. Trickster often becomes unavailable due to Prometheus issues, since its `readinessProbe` depends on Prometheus being accessible.

     Make sure Prometheus is running:

     ```
     d8 k -n d8-monitoring describe pod -l app.kubernetes.io/name=prometheus,prometheus=main
     ```

- DeckhouseModuleUseEmptyDir CE S9
  ▼ **Deckhouse module MODULE_NAME is using emptyDir for storage.**

  The Deckhouse module `MODULE_NAME` is using `emptyDir` as its storage.

  If the associated Pod is removed from the node for any reason, the data in the `emptyDir` is deleted permanently. Consider using persistent storage if data durability is critical for the module.

- GrafanaDashboardAlertRulesDeprecated CE S8
  ▼ **Deprecated Grafana alerts detected.**

  Before upgrading to Grafana 10, migrate outdated alerts from Grafana to an external alertmanager (or `exporter-alertmanager` stack).

  To list all deprecated alert rules, use the following expression:

  ```
  sum by (dashboard, panel, alert_rule)
  (d8_grafana_dashboards_deprecated_alert_rule) > 0
  ```

  Note that this check runs once per hour, so the alert may take up to an hour to clear after deprecated resources are migrated.

- GrafanaDashboardPanelIntervalDeprecated CE S8
  ▼ **Deprecated Grafana panel intervals detected.**

  Before upgrading to Grafana 10, update outdated panel expressions that use `$interval_rv`, `interval_sx3`, or `interval_sx4` to the new variable `$__rate_interval`.

  To list all deprecated panel intervals, use the following expression:

  ```
  sum by (dashboard, panel, interval) (d8_grafana_dashboards_deprecated_interval) >
  0
  ```

  Note that this check runs once per hour, so the alert may take up to an hour to clear after deprecated intervals are removed.

- GrafanaDashboardPluginsDeprecated CE S8
  ▼ **Deprecated Grafana plugins detected.**

Before upgrading to Grafana 10, make sure that the currently installed plugins will work correctly with Grafana 10.

To list all potentially outdated plugins, use the following expression:

```
sum by (dashboard, panel, plugin) (d8_grafana_dashboards_deprecated_plugin) > 0
```

The `flant-statusmap-panel` plugin is deprecated and no longer supported. It's recommended that you migrate to the state timeline plugin instead.

Note that this check runs once per hour, so the alert may take up to an hour to clear after deprecated resources are migrated.

- K8STooManyNodes CE S7
  ▼ Node count is approaching the maximum allowed.

  The cluster is currently running VALUE nodes, which is close to the maximum of XXX allowed nodes.

- PrometheusDirectAccessDeprecated CE S6
  ▼ Deprecated Ingress with direct Prometheus access found in cluster.

  The DKP cluster has an Ingress that allows direct access to Prometheus with client certificate authentication, for example, for the external metrics access. This approach is deprecated and the direct access to Prometheus will not be possible in the future DKP releases.

  To resolve this issue, remove the deprecated Ingress and use the `observability` module to manage Prometheus metrics access.

  Refer to the module documentation for instructions.

- PrometheusDiskUsage CE S4
  ▼ Prometheus disk usage exceeds 95%.

  Deckhouse has detected that the Prometheus disk is over 95% full.

  To check the current usage, run the following command:

  ```
  d8 k -n NAMESPACE exec -ti POD_NAME -c prometheus -- df -PBG /prometheus
  ```

  Consider expanding disk size following the guide.

- PrometheusLongtermRotatingEarlierThanConfiguredRetentionDays CE S4
  ▼ Prometheus-longterm data is rotating earlier than the configured retention period.

  The `prometheus-longterm` instance is rotating data earlier than specified by the `longtermRetentionDays` parameter.

  Troubleshooting options:

  - Increase the Prometheus disk size.
  - Reduce the number of metrics collected.
  - Lower the `longtermRetentionDays` parameter in the module configuration.
- PrometheusMainRotatingEarlierThanConfiguredRetentionDays CE S4
  ▼ Prometheus-main data is rotating earlier than the configured retention period.

  The `prometheus-main` instance is rotating data earlier than specified by the `retentionDays` parameter.

  Troubleshooting options:

  - Increase the Prometheus disk size.
  - Reduce the number of metrics collected.
  - Lower the `retentionDays` parameter in the module configuration.
- PrometheusRemoteWriteEndpointUnavailable CE S7
  ▼ Prometheus remote write endpoint is unavailable.

  Prometheus cannot send metrics to the remote write endpoint `XXX`.

  To troubleshoot you can:

  1. Get list of all Prometheus pods:

     ```
     d8 k -n d8-monitoring get pods -l "app.kubernetes.io/name=prometheus"
     ```

Review logs of each prometheus pod to find the detailed error information:

```
d8 k -n NAMESPACE logs `pod-name` -f
```

2. Get logs from all Prometheus pods to find detailed information:

```
d8 k -n NAMESPACE logs -l "app.kubernetes.io/name=prometheus"
```

- PrometheusScapeConfigDeclarationDeprecated CE S8
  ▾ **AdditionalScrapeConfigs from Secrets will be deprecated soon.**

  Defining additional scrape configurations using Secrets will be deprecated in `prometheus-operator` versions later than v0.65.1.

  Migrate to using the ScrapeConfig custom resource instead.

- PrometheusServiceMonitorDeprecated CE S8
  ▾ **Deprecated Prometheus ServiceMonitor detected.**

  The Kubernetes cluster uses the modern `EndpointSlice` network mechanism, but the ServiceMonitor `NAMESPACE/NAME` still uses relabeling rules based on the old `Endpoints` mechanism (starting with `__meta_kubernetes_endpoints_`).

  Support for the old relabeling rules will be removed in a future Deckhouse release (1.60). To avoid issues, update your ServiceMonitor to use `EndpointSlice`-based relabeling.

  Update relabeling rules as follows:

```
__meta_kubernetes_endpoints_name -> __meta_kubernetes_endpointslice_name
__meta_kubernetes_endpoints_label_XXX -> __meta_kubernetes_endpointslice_label_XXX
__meta_kubernetes_endpoints_labelpresent_XXX -> __meta_kubernetes_endpointslice_labelpresent_XXX
__meta_kubernetes_endpoints_annotation_XXX -> __meta_kubernetes_endpointslice_annotation_XXX
__meta_kubernetes_endpoints_annotationpresent_XXX ->
__meta_kubernetes_endpointslice_annotationpresent_XXX
__meta_kubernetes_endpoint_node_name ->
__meta_kubernetes_endpointslice_endpoint_topology_kubernetes_io_hostname
__meta_kubernetes_endpoint_ready -> __meta_kubernetes_endpointslice_endpoint_conditions_ready
__meta_kubernetes_endpoint_port_name -> __meta_kubernetes_endpointslice_port_name
__meta_kubernetes_endpoint_port_protocol -> __meta_kubernetes_endpointslice_port_protocol
__meta_kubernetes_endpoint_address_target_kind ->
__meta_kubernetes_endpointslice_address_target_kind
__meta_kubernetes_endpoint_address_target_name ->
__meta_kubernetes_endpointslice_address_target_name
```

- TargetDown CE S5
  ▾ **A target is down.**

  Deckhouse has detected that the `JOB_NAME` target is is currently unreachable.

- TargetDown CE S6
  ▾ **A target is down.**

  Deckhouse has detected that the `JOB_NAME` target is is currently unreachable.

- TargetDown CE S7
  ▾ **A target is down.**

  Deckhouse has detected that the `JOB_NAME` target is is currently unreachable.

- TargetSampleLimitExceeded CE S6
  ▾ **Scrapes are exceeding the sample limit.**

  One or more targets are down because the Prometheus sample limit was exceeded during a scrape.

- TargetSampleLimitExceeded CE S7
  ▾ **Sampling limit is nearly reached.**

  The target is close to exceeding the Prometheus’s sampling limit. Less than 10% of the allowed samples are left.

Module secret-copier

- D8SecretCopierDeprecatedLabels CE S9
  ▾ **Obsolete antiopa_secret_copier=yes label detected.**

The `secret-copier` module has changed its labeling approach for original secrets in the `default` namespace.

The label `antiopa-secret-copier: "yes"` is deprecated and will be removed soon.

To resolve this issue, replace the label `antiopa-secret-copier: "yes"` with `secret-copier.deckhouse.io/enabled: ""` for all secrets used by the `secret-copier` module in the `default` namespace.

Module terraform-manager

- D8TerraformStateExporterClusterStateChanged CE S8
  ▼ **Terraform-state-exporter cluster state change detected.**

  The current Kubernetes cluster state is `STATUS_REFERENCE` compared to the Terraform state.

  It is important to reconcile the states.

  Troubleshooting steps:

  1. View the differences:

     ```
     dhctl terraform check
     ```

  2. Apply the necessary changes to bring the cluster in sync:

     ```
     dhctl converge
     ```

- D8TerraformStateExporterClusterStateError CE S8
  ▼ **Terraform-state-exporter cluster state error.**

  The `terraform-state-exporter` can’t check difference between the Kubernetes cluster state and the Terraform state.

  That was likely caused by `terraform-state-exporter` failing to run Terraform with the current state and configuration.

  Troubleshooting steps:

  1. View the differences:

     ```
     dhctl terraform check
     ```

  2. Apply the necessary changes to bring the cluster in sync:

     ```
     dhctl converge
     ```

- D8TerraformStateExporterHasErrors CE S8
  ▼ **Terraform-state-exporter has encountered errors.**

  Errors occurred during the operation of the `terraform-state-exporter`.

  To get more details, check the Pod logs:

  ```
  d8 k -n d8-system logs -l app=terraform-state-exporter -c exporter
  ```

- D8TerraformStateExporterNodeStateChanged CE S8
  ▼ **Terraform-state-exporter node state change detected.**

  The current state of node `NODE_GROUP_NAME/NAME` is `STATUS_REFERENCE` compared to the Terraform state.

  It is important to reconcile the states.

  Troubleshooting steps:

  1. View the differences:

     ```
     dhctl terraform check
     ```

  2. Apply the necessary changes to bring the cluster in sync:

```
dhctl converge
```

- D8TerraformStateExporterNodeStateError CE S8
  ▼ **Terraform-state-exporter node state error.**

  The `terraform-state-exporter` can’t check the difference between the node `NODE_GROUP_NAME/NAME` state and the Terraform state.

  Probably, it occurred because `terraform-manager` had failed to run Terraform with the current state and configuration.

  Troubleshooting steps:

  1. View the differences:

     ```
     dhctl terraform check
     ```

  2. Apply the necessary changes to bring the cluster in sync:

     ```
     dhctl converge
     ```

- D8TerraformStateExporterNodeTemplateChanged CE S8
  ▼ **Terraform-state-exporter node template change detected.**

  The `terraform-state-exporter` has detected a mismatch between the node template in the cluster provider configuration and the one specified in the NodeGroup NAME`.

  Node template is `STATUS_REFERENCE`.

  Troubleshooting steps:

  1. View the differences:

     ```
     dhctl terraform check
     ```

  2. Adjust NodeGroup settings to fix the issue or bring the cluster in sync via the following command:

     ```
     dhctl converge
     ```

- D8TerraformStateExporterPodIsNotReady CE S8
  ▼ **Terraform-state-exporter Pod is not Ready.**

  The `terraform-state-exporter` cannot check the difference between the actual Kubernetes cluster state and the Terraform state.

  To resolve the issue, check the following:

  1. Deployment description:

     ```
     d8 k -n d8-system describe deployment terraform-state-exporter
     ```

  2. Pod status:

     ```
     d8 k -n d8-system describe pod -l app=terraform-state-exporter
     ```

- D8TerraformStateExporterPodIsNotRunning CE S8
  ▼ **Terraform-state-exporter Pod is not Running.**

  The `terraform-state-exporter` cannot check the difference between the actual Kubernetes cluster state and the Terraform state.

  To resolve the issue, check the following:

  1. Deployment description:

     ```
     d8 k -n d8-system describe deployment terraform-state-exporter
     ```

  2. Pod status:

```
d8 k -n d8-system describe pod -l app=terraform-state-exporter
```

- D8TerraformStateExporterTargetAbsent CE S8
  ▼ **Terraform-state-exporter target is missing in Prometheus.**

  Prometheus cannot find the `terraform-state-exporter` target.

  To investigate the details:

  - Check the Pod status:

    ```
    d8 k -n d8-system get pod -l app=terraform-state-exporter
    ```

  - Check the container logs:

    ```
    d8 k -n d8-system logs -l app=terraform-state-exporter -c exporter
    ```

- D8TerraformStateExporterTargetDown CE S8
  ▼ **Prometheus can't scrape terraform-state-exporter.**

  Prometheus is unable to scrape metrics from the `terraform-state-exporter`.

  To investigate the details:

  - Check the Pod status:

    ```
    d8 k -n d8-system get pod -l app=terraform-state-exporter
    ```

  - Check the container logs:

    ```
    d8 k -n d8-system logs -l app=terraform-state-exporter -c exporter
    ```

Module upmeter

- D8SmokeMiniNotBoundPersistentVolumeClaims CE S9
  ▼ **Smoke-mini has unbound or lost PersistentVolumeClaims.**

  The PersistentVolumeClaim `PVC_NAME` is in the `STATUS` phase.

  This indicates a problem with PersistentVolume provisioning.

  To investigate the cause, check the PersistentVolumeClaim phase:

  ```
  d8 k -n d8-upmeter get pvc PVC_NAME
  ```

  If your cluster doesn’t have a disk provisioning system, you can disable volume ordering for `smoke-mini` using the module settings.

- D8UpmeterAgentPodIsNotReady CE S6

  Upmeter agent is not Ready.

- D8UpmeterAgentReplicasUnavailable CE S6
  ▼ **One or more upmeter-agent Pods are NOT Running.**

  Some `upmeter-agent` Pods are not in the Running state.

  To investigate the details, do the following:

  - Check the DaemonSet status:

    ```
    d8 k -n d8-upmeter get daemonset upmeter-agent -o json | jq .status
    ```

  - Check the Pod status:

    ```
    d8 k -n d8-upmeter get pods -l app=upmeter-agent -o json | jq '.items[] |
    {(.metadata.name):.status}'
    ```

- D8UpmeterDiskUsage CE S5
  ▼ **Upmeter disk usage exceeds 80%.**

  Disk usage for Upmeter has exceeded 80%.

  The only way to resolve this is to recreate the PersistentVolumeClaim (PVC) in the following steps:

  1. Save the PVC data if you need it.
  2. Delete the PVC and restart the `upmeter` Pod:

     ```
     d8 k -n d8-upmeter delete persistentvolumeclaim/data-upmeter-0 pod/upmeter-0
     ```

  3. Check the status of the created PVC:

     ```
     d8 k -n d8-upmeter get pvc
     ```

- D8UpmeterProbeGarbageConfigmap CE S9
  ▼ **Garbage ConfigMaps from the basic probe are not being cleaned up.**

  Probe-generated ConfigMaps were found but not cleaned up as expected.

  `upmeter-agent` should automatically delete ConfigMaps produced by the `control-plane/basic` probe. There should be no more ConfigMaps than there are master nodes (since `upmeter-agent` runs as a DaemonSet with a master `nodeSelector`), and they should be deleted within seconds.

  This may indicate:

  - A problem with the `kube-apiserver`.
  - Stale ConfigMaps left by outdated `upmeter-agent` Pods after an Upmeter update.

  Troubleshooting steps:

  1. Check the `upmeter-agent` logs:

     ```
     d8 k -n d8-upmeter logs -l app=upmeter-agent --tail=-1 | jq -rR 'fromjson? |
     select(.group=="control-plane" and .probe == "basic-functionality") |
     [.time, .level, .msg] | @tsv'
     ```

  2. Ensure the control plane is operating normally.
  3. Delete stale ConfigMaps manually:

     ```
     d8 k -n d8-upmeter delete cm -l heritage=upmeter
     ```

- D8UpmeterProbeGarbageDeployment CE S9
  ▼ **Garbage Deployments from the controller-manager probe are not being cleaned up.**

  The average number of probe-generated Deployments per `upmeter-agent` Pod is VALUE.

  `upmeter-agent` should automatically delete Deployments created by the `control-plane/controller-manager` probe. There should not be more Deployments than master nodes (since `upmeter-agent` runs as a DaemonSet with a master `nodeSelector`), and they should be deleted within seconds.

  This may indicate:

  - A problem with the `kube-apiserver`.
  - Stale Deployments left by outdated `upmeter-agent` Pods after an Upmeter update.

  Troubleshooting steps:

  1. Check the `upmeter-agent` logs:

     ```
     d8 k -n d8-upmeter logs -l app=upmeter-agent --tail=-1 | jq -rR 'fromjson? |
     select(.group=="control-plane" and .probe == "controller-manager") |
     [.time, .level, .msg] | @tsv'
     ```

  2. Ensure the control plane is operating normally. Pay close attention to `kube-controller-manager`.

3. Delete stale Deployments manually:

```
d8 k -n d8-upmeter delete deployment -l heritage=upmeter
```

- D8UpmeterProbeGarbageNamespaces CE S9
  ▼ Garbage namespaces from the namespace probe are not being cleaned up.

  The average number of probe-created namespaces per `upmeter-agent` Pod is VALUE.

  `upmeter-agent` should automatically clean up namespaces created by the `control-plane/namespace` probe.
  There should not be more of these namespaces than there are master nodes (since `upmeter-agent` runs as a DaemonSet with a master `nodeSelector`), and they should be deleted within seconds.

  This may indicate:

  - A problem with the `kube-apiserver`.
  - Stale namespaces left from older `upmeter-agent` Pods after an Upmeter update.

  Troubleshooting steps:

  1. Check `upmeter-agent` logs:

     ```
     d8 k -n d8-upmeter logs -l app=upmeter-agent --tail=-1 | jq -rR 'fromjson? |
     select(.group=="control-plane" and .probe == "namespace") | [.time, .level,
     .msg] | @tsv'
     ```

  2. Ensure the control plane is operating normally.

  3. Delete stale namespaces manually:

     ```
     d8 k -n d8-upmeter delete ns -l heritage=upmeter
     ```

- D8UpmeterProbeGarbagePods CE S9
  ▼ Garbage Pods from the scheduler probe are not being cleaned up.

  The average number of probe Pods per `upmeter-agent` Pod is VALUE.

  `upmeter-agent` should automatically clean up Pods created by the `control-plane/scheduler` probe.
  There should not be more of these Pods than there are master nodes (since `upmeter-agent` runs as a DaemonSet with a master `nodeSelector`), and they should be deleted within seconds.

  This may indicate:

  - A problem with the `kube-apiserver`.
  - Stale Pods left from old `upmeter-agent` Pods after an Upmeter update.

  Troubleshooting steps:

  1. Check `upmeter-agent` logs:

     ```
     d8 k -n d8-upmeter logs -l app=upmeter-agent --tail=-1 | jq -rR 'fromjson? |
     select(.group=="control-plane" and .probe == "scheduler") | [.time, .level,
     .msg] | @tsv'
     ```

  2. Ensure the control plane is operating normally.

  3. Delete stale Pods manually:

     ```
     d8 k -n d8-upmeter delete pods -l upmeter-probe=scheduler
     ```

- D8UpmeterProbeGarbagePodsFromDeployments CE S9
  ▼ Garbage Pods from the controller-manager probe are not being cleaned up.

  A Pod from the `control-plane/controller-manager` probe was created more than 5 minutes ago.

  `upmeter-agent` is expected to clean Deployments created by the `control-plane/controller-manager` probe, and `kube-controller-manager` should remove the associated Pods within

seconds.

This may indicate:

- A problem with the `kube-apiserver` or `kube-controller-manager`.
- Stale Pods left from outdated `upmeter-agent` Pods after an Upmeter update.

Troubleshooting steps:

1. Check the `upmeter-agent` logs:

   ```
   d8 k -n d8-upmeter logs -l app=upmeter-agent --tail=-1 | jq -rR 'fromjson? |
   select(.group=="control-plane" and .probe == "controller-manager") |
   [.time, .level, .msg] | @tsv'
   ```

2. Ensure the control plane is operating normally. Pay close attention to `kube-controller-manager`.
3. Delete stale Pods manually:

   ```
   d8 k -n d8-upmeter delete pods -l upmeter-probe=controller-manager
   ```

- D8UpmeterProbeGarbageSecretsByCertManager CE S9
  ▼ Garbage Secrets from the cert-manager probe are not being cleaned up.

  Probe-generated Secrets were found.

  `upmeter-agent` should clean up certificates created by the `cert-manager` probe, and `cert-manager` in turn should clean up the associated Secrets. There should not be more of these Secrets than there are master nodes (since `upmeter-agent` runs as a DaemonSet with a master `nodeSelector`), and they should be deleted within seconds.

  This may indicate:

  - A problem with `kube-apiserver`, `cert-manager`, or `upmeter`.
  - Stale Secrets left from older `upmeter-agent` Pods after an Upmeter update.

  Troubleshooting steps:

  1. Check `upmeter-agent` logs:

     ```
     d8 k -n d8-upmeter logs -l app=upmeter-agent --tail=-1 | jq -rR 'fromjson? |
     select(.group=="control-plane" and .probe == "cert-manager") | [.time,
     .level, .msg] | @tsv'
     ```

  2. Check that the control plane and `cert-manager` are operating normally.
  3. Delete stale certificates and Secrets manually:

     ```
     d8 k -n d8-upmeter delete certificate -l upmeter-probe=cert-manager
     d8 k -n d8-upmeter get secret -ojson | jq -r '.items[] | .metadata.name' | grep
     upmeter-cm-probe | xargs -n 1 -- d8 k -n d8-upmeter delete secret
     ```

- D8UpmeterServerPodIsNotReady CE S6

  Upmeter server is not Ready.

- D8UpmeterServerPodIsRestartingTooOften CE S9
  ▼ Upmeter server is restarting too frequently.

  The `upmeter` server has restarted VALUE times in the last hour.

  Frequent restarts may indicate an issue. It is expected to run continuously and collect availability episodes without interruption.

  To investigate the cause, check the logs:

  ```
  d8 k -n d8-upmeter logs -f upmeter-0 upmeter
  ```

- D8UpmeterServerReplicasUnavailable CE S6
  ▼ One or more Upmeter server Pods are NOT Running.

Some `upmeter` Pods are not in the Running state.

To investigate the details, do the following:

- Check the StatefulSet status:

```
d8 k -n d8-upmeter get statefulset upmeter -o json | jq .status
```

- Check the Pod status:

```
d8 k -n d8-upmeter get pods upmeter-0 -o json | jq '.items[] |
{(.metadata.name):.status}'
```

- D8UpmeterSmokeMiniMoreThanOnePVxPVC CE S9
  ▼ **Unnecessary smoke-mini PersistentVolumes detected in the cluster.**

  The number of unnecessary `smoke-mini` PersistentVolumes (PVs) is VALUE.

  These PVs should be deleted automatically when released.

  Possible causes:

  - The `smoke-mini` StorageClass may be set to `Retain` by default.
  - There may be issues with the CSI driver or cloud storage integration.

  These PVs do not contain valuable data and should be deleted.

  To list all the PVs, run the following command:

```
d8 k get pv | grep disk-smoke-mini
```

- D8UpmeterTooManyHookProbeObjects CE S9
  ▼ **Too many UpmeterHookProbe objects in the cluster.**

  The average number of `UpmeterHookProbe` objects per `upmeter-agent` Pod is VALUE, but it should always be exactly 1 per agent.

  This likely happened because older `upmeter-agent` Pods left behind their `UpmeterHookProbe` objects during an Upmeter update or downscale.

  Once the cause has been investigated, remove outdated objects and leave only the ones corresponding to currently running `upmeter-agent` Pods.

  To view all `UpmeterHookProbe` objects, run the following command:

```
d8 k get upmeterhookprobes.deckhouse.io
```

Module user-authn

- D8DexAllTargetsDown CE S6

  Prometheus is unable to scrape Dex metrics.

- D8DexAuthRequestsQuotaExhausted CE S3
  ▼ **Dex AuthRequest objects ResourceQuota is exhausted in namespace "d8-user-authn".**

  The ResourceQuota for `authrequests.dex.coreos.com` objects in namespace `d8-user-authn` has been fully exhausted.

  Dex is currently unable to create new AuthRequest custom resources, which will cause:

  - Dex health checks to fail immediately.
  - Authentication through external identity providers to stop working.
  - Users unable to log in via OAuth/OIDC flows.

  Immediate actions required:

  1. Check current AuthRequest objects and clean up stale ones:

```
d8 k get authrequests.dex.coreos.com -n d8-user-authn --no-headers | wc -l
d8 k delete authrequests.dex.coreos.com --all -n d8-user-authn
```

2. Temporarily increase the ResourceQuota limit:

   ```
   d8 k patch resourcequota authrequest-quota -n d8-user-authn --type='json' -p='[{"op":
   "replace", "path": "/spec/hard/count\/authrequests.dex.coreos.com",
   "value": "20000"}]'
   ```

3. Investigate why Dex is not cleaning up AuthRequest objects automatically.
4. Review Dex configuration for AuthRequest TTL settings.

- D8DexAuthRequestsQuotaHigh CE S4
  ▼ **Dex AuthRequest objects quota usage is higher than 80% in namespace "d8-user-authn".**

  The ResourceQuota for `authrequests.dex.coreos.com` objects in namespace `d8-user-authn` is approaching its limit.

  Current usage: XXX%

  When the quota is exhausted, Dex will be unable to create new AuthRequest custom resources, which may cause:

  - Dex health checks to fail.
  - Authentication through external identity providers (GitLab, GitHub, etc.) to stop working.

  Investigation steps:

  1. Check for stale AuthRequest objects that Dex should have cleaned up:

     ```
     d8 k get authrequests.dex.coreos.com -n d8-user-authn
     ```

  2. Review Dex configuration for AuthRequest TTL settings.
  3. Consider temporarily increasing the ResourceQuota limit if needed.

Sysctl parameters managed by the platform

Deckhouse automatically configures and manages a set of the server’s kernel parameters using the `sysctl` utility. The configured parameters improve network throughput, prevent resource depletion, and optimize memory management.

> **i** If you modify these parameters, Deckhouse will automatically revert them to the values listed below.

Parameter	Value set by Deckhouse	Description
`/sys/block/*/queue/nr_requests`	`256`	Number of queued requests for block devices.
`/sys/block/*/queue/read_ahead_kb`	`256`	Amount of extra data that kernel reads from the disk to improve future read performance.
`/sys/kernel/mm/transparent_hugepage/enabled`	`never`	Disables Transparent HugePage.
`/sys/kernel/mm/transparent_hugepage/defrag`	`never`	Disables the Transparent HugePage defragmentation.
`/sys/kernel/mm/transparent_hugepage/use_zero_page`	`0`	Disables usage of huge zero pages.
`/sys/kernel/mm/transparent_hugepage/khugepaged/defrag`	`0`	Disables `khugepaged` defragmentation.
`/proc/sys/net/ipv4/conf/*/rp_filter`	`0`	Disables reverse path filtering for all interfaces.
`fs.file-max`	`1000000`	Maximum number of open files.

Parameter	Value set by Deckhouse	Description
`fs.inotify.max_user_instances`	`5120`	Maximum number of inotify instances.
`fs.inotify.max_user_watches`	`524288`	Maximum number of files monitored by a single inotify instance.
`fs.may_detach_mounts`	`1`	Allows lazy unmounting of a file system.
`kernel.numa_balancing`	`0`	Disables automatic NUMA memory balancing.
`kernel.panic`	`10 (0 if fencing is enabled)`	Time in seconds until the node reboots after it encounters the fatal kernel panic error. By default, it's set to `10`. If `fencing` mode is enabled for the node, it's set to `0`, preventing the node from rebooting.
`kernel.panic_on_oops`	`1`	Allows the system to trigger a kernel panic after an unexpected oops error. Required for kubelet to work correctly.
`kernel.pid_max`	`2000000`	Maximum number of process IDs that can be assigned in the system.
`net.bridge.bridge-nf-call-arptables`	`1`	Enables traffic filtering through arptables. Required for kube-proxy to work correctly.
`net.bridge.bridge-nf-call-ip6tables`	`1`	Enables traffic filtering through ip6tables. Required for kube-proxy to work correctly.
`net.bridge.bridge-nf-call-iptables`	`1`	Enables traffic filtering through iptables. Required for kube-proxy to work correctly.
`net.core.netdev_max_backlog`	`5000`	Maximum number of packets allowed in the processing queue.
`net.core.rmem_max`	`16777216`	Maximum receive buffer size in bytes.
`net.core.somaxconn`	`1000`	Maximum number of pending connections.
`net.core.wmem_max`	`16777216`	Maximum send buffer size in bytes.
`net.ipv4.conf.all.forwarding`	`1`	Enables IPv4 packet forwarding between network interfaces. Equivalent to the `net.ipv4.ip_forward` parameter.
`net.ipv4.ip_local_port_range`	`"32768 61000"`	Range of ports available for outgoing TCP and UDP connections.
`net.ipv4.neigh.default.gc_thresh1`	`16384`	Lower threshold for the amount of ARP entries after which the system starts cleaning up old entries.

Parameter	Value set by Deckhouse	Description
`net.ipv4.neigh.default.gc_thresh2`	`28672`	Middle threshold for the amount of ARP entries after which the system starts garbage collection.
`net.ipv4.neigh.default.gc_thresh3`	`32768`	Absolute maximum number of ARP entries.
`net.ipv4.tcp_max_syn_backlog`	`8096`	Maximum number of queued SYN connections.
`net.ipv4.tcp_no_metrics_save`	`1`	Disables saving of TCP metrics of closed connections and reusing them for new connections.
`net.ipv4.tcp_rmem`	`"4096 12582912 16777216"`	Receive buffer sizes for incoming TCP packets in bytes: `"<minimum> <default> <maximum>"`.
`net.ipv4.tcp_slow_start_after_idle`	`0`	Disables using the congestion window (CWND) and slow start algorithm for TCP connections.
`net.ipv4.tcp_tw_reuse`	`1`	Enables reusing the outgoing TCP connections in `TIME-WAIT` state.
`net.ipv4.tcp_wmem`	`"4096 12582912 16777216"`	Send buffer sizes for outgoing TCP packets in bytes: `"<minimum> <default> <maximum>"`.
`net.netfilter.nf_conntrack_max`	`<no-of-cores * 131072> or 524288`	Maximum number of tracked connections in the conntrack table. Calculated as “number of CPU cores” * 131072, but no lower than `524288`.
`net.nf_conntrack_max`	`<no-of-cores * 131072> or 524288`	Maximum number of tracked connections in the conntrack table for older kernels. Calculated as “number of CPU cores” * 131072, but no lower than `524288`.
`vm.dirty_background_ratio`	`5`	Percentage of system memory that can be filled with dirty pages before the kernel starts writing them to disk in the background.
`vm.dirty_expire_centisecs`	`12000`	Duration (in centiseconds) a dirty page can remain in system memory before it must be written to disk.
`vm.dirty_ratio`	`80`	Percentage of system memory that can be filled with dirty pages before all processes must stop and flush data to disk.
`vm.min_free_kbytes`	`131072`	Minimum amount of free memory in kilobytes reserved by the kernel for critical operations.
`vm.overcommit_memory`	`1`	Enables memory overcommitment.
`vm.swappiness`	`0`	Disables swap file usage.

Additional resources

- Module `node-manager` custom resources

Information about used software

List of third-party software used in the Deckhouse Kubernetes Platform. Includes **only built-in modules** (doesn't include modules from sources).

The list of third-party software used in a specific module can be found on the overview page in the module documentation.

cert-manager

Version: 1.20.0

A Kubernetes add-on to automate the management and issuance of TLS certificates from various issuing sources.

Used in module cert-manager

Apache License 2.0

Chrony
chrony is a versatile implementation of the Network Time Protocol (NTP)

Used in module chrony

GNU General Public License version 2

Cilium

Version: 1.17.4

Cilium is open source software for providing and transparently securing network connectivity and loadbalancing between application workloads such as application containers or processes.

Used in module cni-cilium

Apache License 2.0

Cluster API

Version: 1.11.3

Kubernetes Cluster API for declarative cluster lifecycle management.

Used in module node-manager

Apache License 2.0

Cluster Autoscaler
Kubernetes autoscaler that automatically adjusts the size of the cluster.

Used in module node-manager

Apache License 2.0

CNI plugins

Version: 1.7.1

Some reference and example networking plugins, maintained by the CNI team.

Used in module cni-cilium

Apache License 2.0

CoreDNS
A DNS server/forwarder that chains plugins.

Used in module kube-dns

Apache License 2.0

Descheduler

Version: 0.35.1

A tool to auto-fix scheduling issues Kubernetes does not cover itself.

Used in module descheduler

Apache License 2.0

Dex

Version: 2.44.0

A Federated OpenID Connect Provider with pluggable connectors.

Used in module user-authn

Apache License 2.0

Etcd

Version: 3.6.10

A distributed reliable key-value store for the most critical data of a distributed system.

Used in module control-plane-manager

Apache License 2.0

Flannel

Version: 0.27.4

A simple and easy way to configure a layer 3 network fabric designed for Kubernetes.

Used in module cni-flannel

Apache License 2.0

Gardener Machine Controller Manager

Version: v0.36.0

An operator that manages VMs as a kubernetes custom resource.

Used in module node-manager

Apache License 2.0

Gatekeeper

Version: 3.22.0

Policy-based control for cloud native environments

Used in module admission-policy-engine

Apache License 2.0

Gops

Version: 0.3.27

A tool to list and diagnose Go processes currently running on your system.

Used in module cni-cilium

BSD 3-Clause License

Grafana
An open-source platform for monitoring and observability.

Used in module prometheus

GNU Affero General Public License v3.0

Hubble UI

Version: 0.13.2

Hubble UI is an open-source user interface for Cilium Hubble.

Used in module cilium-hubble

Apache License 2.0

Ingress NGINX Controller

Versions: 1.10.4, 1.12.1, 1.14.4

A ingress controller for Kubernetes using NGINX as a reverse proxy and load balancer.

Used in module ingress-nginx

Apache License 2.0

Istio

Versions: 1.21.6, 1.25.2

An open platform to connect, manage, and secure microservices.

Used in module istio

Apache License 2.0

JQ

Version: 1.7.1

A lightweight and flexible command-line JSON processor.

Used in module deckhouse

Mixed (https://github.com/stedolan/jq/blob/master/COPYING)

keepalived

Version: 2.2.7

Provides simple and robust facilities for loadbalancing and high-availability to Linux system and Linux based infrastructures.

Used in module keepalived

GNU General Public License v2.0

Kiali

Versions: 1.81.0, 2.7.1

Visualisation tool for the istio service mesh topology, and features like circuit breakers or request rates.

Used in module istio

Apache License 2.0

Kube-router

Version: 2.0.1

A turnkey solution for Kubernetes networking with aim to provide operational simplicity and high performance.

Used in module network-policy-engine

Apache License 2.0

kube-state-metrics
A simple service that listens to the Kubernetes API server and generates metrics about the state of the objects.

Used in module monitoring-kubernetes

Apache License 2.0

Kubernetes
An open source system for managing containerized applications across multiple hosts.

Used in module control-plane-manager

Apache License 2.0

Kubernetes Autoscaler

Version: 1.6.0

A autoscaling-related components for Kubernetes.

Used in module vertical-pod-autoscaler

Apache License 2.0

Kubernetes Metrics Server
A scalable, efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines.

Used in module prometheus-metrics-adapter

Apache License 2.0

Libcap-ng

Version: 0.8.5

Libcap-ng is a library for Linux that makes using posix capabilities easy..

Used in module openvpn

GNU General Public License v2.0

Local-path-provisioner
Local Path Provisioner provides a way for the Kubernetes users to utilize the local storage in each node by creating hostPath based persistent volume on the node automatically.

Used in module local-path-provisioner

Apache License 2.0

Loki
Loki: like Prometheus, but for logs.

Used in module loki

GNU Affero General Public License v3.0

Memcached
High performance multithreaded event-based key/value cache store intended to be used in a distributed system.

Used in module prometheus

BSD-3-Clause license

MetalLB

Version: 0.15.2

A load-balancer implementation for bare metal Kubernetes clusters, using standard routing protocols.

Used in module metallb

Apache License 2.0

Mimir
A HTTP reverse proxy/cache for http applications and a dashboard query accelerator for time series databases.

Used in module prometheus

AGPL-3.0 license

ModSecurity

Version: 3.0.14

ModSecurity is an open source, cross platform web application firewall (WAF) engine for Apache, IIS and Nginx.

Used in module ingress-nginx

Apache License 2.0

nlohmann\json

nlohmann/json

Version: 3.11.3

JSON for Modern C++

Used in module ingress-nginx

MIT License

Node exporter
Prometheus exporter for hardware and OS metrics exposed by *NIX kernels, written in Go with pluggable metric collectors.

Used in module monitoring-kubernetes

Apache License 2.0

OAuth2 Proxy

Version: 7.5.1

A reverse proxy and static file server that provides authentication using Providers (Google, GitHub, and others) to validate accounts by email, domain or group.

Used in module user-authn

MIT License

OpenKruise

Version: 898dc482a0cbf4fa0db214f952319aa5e1229152

OpenKruise is an extended component suite for Kubernetes, which mainly focuses on application automations, such as deployment, upgrade, ops and availability protection.

Used in module ingress-nginx

Apache License 2.0

OpenSSL

Version: 3.3.2

TLS/SSL and crypto library.

Used in module openvpn

Apache License 2.0

OpenVPN

Version: 2.6.12

A secure tunneling daemon.

Used in module openvpn

Mixed

OWASP CRS

Version: 4.10.0

The OWASP CRS is a set of generic attack detection rules for use with ModSecurity or compatible web application firewalls.

Used in module ingress-nginx

Apache License 2.0

Prometheus
A systems and service monitoring system.

Used in module prometheus

Apache License 2.0

Prometheus Operator
Kubernetes native deployment and management of Prometheus and related monitoring components.

Used in module operator-prometheus

Apache License 2.0

Prometheus Pushgateway
A tool to allow ephemeral and batch jobs to expose their metrics to Prometheus.

Used in module prometheus-pushgateway

Apache License 2.0

Promxy
Promxy is a prometheus proxy that makes many shards of prometheus appear as a single API endpoint to the user. This significantly simplifies operations and use of prometheus at scale (when you have more than one prometheus host).

Used in module prometheus

MIT License

Shell operator

Version: 1.9.3

A tool for running event-driven scripts in a Kubernetes cluster

Used in module deckhouse

Apache License 2.0

Terraform
A tool for building, changing, and versioning infrastructure safely and efficiently.

Used in module terraform-manager

Mozilla Public License 2.0

Trickster
A HTTP reverse proxy/cache for http applications and a dashboard query accelerator for time series databases.

Used in module prometheus

Apache License 2.0

Vector
Vector is a high-performance, observability data pipeline for logs and metrics

Used in module log-shipper

Mozilla Public License, version 2.0

Used directors

If Kubernetes cluster nodes are analyzed by security scanners (antivirus tools), you may need to configure them to avoid false positives.

Deckhouse Kubernetes Platform (DKP) uses the following directories when running (download in CSV):

- `/mnt/kubernetes-data` (master node) — only exists in clusters deployed in the cloud when a separate disk is used for the etcd database.
- `/var/lib/etcd` (master node) — etcd database.
- `/var/lib/deckhouse/` (master node) — files of Deckhouse modules, which dynamically loads from a registry.
- `/var/lib/upmeter` (master node, the `upmeter` module) — the upmeter module database.
- `/etc/kubernetes` (any node) — manifests of static pods, PKI certificate files.
- `/var/lib/bashible` (any node) — node configuration files.
- `/var/lib/containerd` (any node) — used to store data related to CRI operation (e.g., containerd); contains container image layers, container file system snapshots, meta information, logs, and other container information.
- `/mnt/vector-data` (any node, the `log-shipper` module) — checkpoints of sent logs.

- `/var/log/containers` (any node) — logs of containers (when using `containerd`).
- `/var/lib/kubelet/` (any node) — `kubelet` configuration files.
- `/opt/cni/bin/` (any node) — CNI plugin executables.
- `/opt/deckhouse/bin/` (any node) — executable files required for Deckhouse to work.
- `/var/log/pods/` (any node) — logs of all pod containers that are running on this cluster node.
- `/etc/cni/` (any node) — CNI plugin configuration files.

Deckhouse CLI overview and installation

Deckhouse CLI (`d8`) is a command-line interface for working with products in the Deckhouse ecosystem. It combines commands for working with Kubernetes, administering Deckhouse Kubernetes Platform, delivering applications, copying images, creating backups, collecting diagnostic information, virtualization, working with user data, and other tasks.

Starting from release 1.59, `d8` is automatically installed on all DKP cluster nodes. The utility can also be installed on a separate administrator machine.

Main command groups

`d8` commands are grouped by purpose:

Commands	Purpose
`d8 k`	Commands for managing a cluster (functionally equivalent to `kubectl`).
`d8 system`	Commands for working with platform system settings, configuration, modules, queues, packages, and system logs.
`d8 status`	Command for obtaining the current platform status.
`d8 backup`	Commands for creating backups of key cluster data and generating diagnostic dumps.
`d8 cr`	Commands for working with the container registry.
`d8 mirror`	Commands for copying the platform component images to the local filesystem or a third-party container registry.
`d8 delivery-kit`	Commands for building, publishing, planning, and deploying applications, as well as for working with Helm, registries, SBOM, and related operations.
`d8 data`	Commands for managing data, as well as exporting and importing it.
`d8 v`	Commands for working with virtual machines created a cluster.
`d8 iam`	Commands for managing Deckhouse users, groups, and access grants.
`d8 stronghold`	Commands for managing the lifecycle of secrets (stronghold module)
`d8 network`	Commands for performing network-related operations in the Deckhouse ecosystem, including CNI migration.
`d8 completion`	Commands for generating shell completion scripts.
`d8 tools`	Auxiliary utilities.
`d8 help`	Built-in help.

How do I install the Deckhouse CLI?

There are two ways to install the Deckhouse CLI:

- Starting with version 0.10, you can install using trdl. This method allows you to get fresh versions of the tool with all improvements and fixes.

  > **i** Note that trdl installation requires Internet access to the tuf repository containing the tool. This method will not work in an air-gapped environment!

- You can manually download the executable file and install it on the system.

trdl-based installation

Starting with the Deckhouse CLI version 0.10, you can install it using trdl.

> ⚠ Versions earlier than 0.10 must be uninstalled before proceeding.
>
> If you need to install one of the versions below 0.10, use the outdated installation method.

1. Install trdl client.
2. Add the Deckhouse CLI repository to trdl:

   ```
   URL=https://deckhouse.ru/downloads/deckhouse-cli-trdl
   ROOT_VERSION=1
   ROOT_SHA512=343bd5f0d8811254e5f0b6fe292372a7b7eda08d276ff255229200f84e58a8151ab2729df3515cb11372dc3
   899c70df172a4e54c8a596a73d67ae790466a0491
   REPO=d8

   trdl add $REPO $URL $ROOT_VERSION $ROOT_SHA512
   ```

3. Install the latest stable release of the `d8` utility and check if it works as expected:

   ```
   . $(trdl use d8 0 stable) && d8 --version
   ```

To avoid having to run `. $(trdl use d8 0 stable)` before every Deckhouse CLI invocation, add the following line to your shell's RC file: `alias d8='trdl exec d8 0 stable -- "$@"'`.

Congratulations, you have installed the Deckhouse CLI.

Installing the executable

- Linux x86-64
- macOS x86-64
- macOS ARM64
- Windows

> ⚠ The commands below apply to Deckhouse CLI version v0.32.5, which is bundled with the Deckhouse Kubernetes Platform. The latest version of Deckhouse CLI is available on GitHub. You can check your current Deckhouse CLI version using the `d8 --version` command.

1. Download the archive for Linux x86-64 in a convenient way:
   - **From your cluster**:
     1. Turn on the module deckhouse-tools;
     2. Open the web interface with download links. The web interface address is generated according to the publicDomainTemplate template of the Deckhouse global configuration parameter (the `%s` key is replaced with `tools`).

        For example, if `publicDomainTemplate` is set as `%s-kube.company.my`, the web interface will be available at `tools-kube.company.my`;

3. Click on the button to download Deckhouse CLI for Linux.

- **From the Deckhouse website**:

```
curl -L0 https://deckhouse.io/downloads/deckhouse-cli/v0.32.5/d8-v0.32.5-linux-amd64.tar.gz
```

2. Extract the archive:

```
tar -xvf "d8-v0.32.5-linux-amd64.tar.gz" "linux-amd64/bin/d8"
```

3. Move the file `d8` to the directory listed in the `PATH` system environment variable (for example, `/usr/local/bin/`):

```
sudo mv "linux-amd64/bin/d8" /usr/local/bin/
```

> **i** `PATH` is a system environment variable containing a list of directories with executable files. This is where the OS will look for the binary when a command is run in the terminal. You can view this list with the `echo $PATH` command.

4. Check that the CLI is working:

```
d8 help
```

Congrats, you have successfully installed Deckhouse CLI.

> ⚠ The commands below apply to Deckhouse CLI version v0.32.5, which is bundled with the Deckhouse Kubernetes Platform. The latest version of Deckhouse CLI is available on GitHub. You can check your current Deckhouse CLI version using the `d8 --version` command.

1. Download the archive for macOS x86-64 in a convenient way:

- **From your cluster**:

1. Turn on the module deckhouse-tools;
2. Open the web interface with download links. The web interface address is generated according to the publicDomainTemplate template of the Deckhouse global configuration parameter (the `%s` key is replaced with `tools`).

   For example, if `publicDomainTemplate` is set as `%s-kube.company.my`, the web interface will be available at `tools-kube.company.my`;
3. Click on the button to download Deckhouse CLI for macOS.

- **From the Deckhouse website**:

```
curl -L0 https://deckhouse.io/downloads/deckhouse-cli/v0.32.5/d8-v0.32.5-darwin-amd64.tar.gz
```

2. Extract the archive:

```
tar -xvf "d8-v0.32.5-darwin-amd64.tar.gz" "darwin-amd64/bin/d8"
```

3. Move the file `d8` to the directory listed in the `PATH` system environment variable (for example, `/usr/local/bin/`):

```
sudo mv "darwin-amd64/bin/d8" /usr/local/bin/
```

> **i** `PATH` is a system environment variable containing a list of directories with executable files. This is where the OS will look for the binary when a command is run in the terminal. You can view this list with the `echo $PATH` command.

4. Check that the CLI is working:

```
d8 help
```

Congrats, you have successfully installed Deckhouse CLI.

⚠ The commands below apply to Deckhouse CLI version v0.32.5, which is bundled with the Deckhouse Kubernetes Platform. The latest version of Deckhouse CLI is available on GitHub. You can check your current Deckhouse CLI version using the `d8 --version` command.

1. Download the archive for macOS ARM64 in a convenient way:
   - **From your cluster**:
     1. Turn on the module deckhouse-tools;
     2. Open the web interface with download links. The web interface address is generated according to the publicDomainTemplate template of the Deckhouse global configuration parameter (the `%s` key is replaced with `tools`).

        For example, if `publicDomainTemplate` is set as `%s-kube.company.my`, the web interface will be available at `tools-kube.company.my`;
     3. Click on the button to download Deckhouse CLI for macOS.
   - **From the Deckhouse website**:

     ```
     curl -LO https://deckhouse.io/downloads/deckhouse-cli/v0.32.5/d8-v0.32.5-darwin-arm64.tar.gz
     ```
2. Extract the archive:

```
tar -xvf "d8-v0.32.5-darwin-arm64.tar.gz" "darwin-arm64/bin/d8"
```

3. Move the file `d8` to the directory listed in the `PATH` system environment variable (for example, `/usr/local/bin/`):

```
sudo mv "darwin-arm64/bin/d8" /usr/local/bin/
```

ℹ `PATH` is a system environment variable containing a list of directories with executable files. This is where the OS will look for the binary when a command is run in the terminal. You can view this list with the `echo $PATH` command.

4. Check that the CLI is working:

```
d8 help
```

Congrats, you have successfully installed Deckhouse CLI.

⚠ The commands below apply to Deckhouse CLI version v0.32.5, which is bundled with the Deckhouse Kubernetes Platform. The latest version of Deckhouse CLI is available on GitHub. You can check your current Deckhouse CLI version using the `d8 --version` command.

1. Download the archive for Windows x86-64 in a convenient way.
   - **From your cluster**:
     1. Turn on the module deckhouse-tools;
     2. Open the web interface with download links. The web interface address is generated according to the publicDomainTemplate template of the Deckhouse global configuration parameter (the `%s` key is replaced with `tools`).

        For example, if `publicDomainTemplate` is set as `%s-kube.company.my`, the web interface will be available at `tools-kube.company.my`;

3. Click on the button to download Deckhouse CLI for Windows.

- **From the Deckhouse website**, download the archive from this link: Windows x86-64.

2. Extract the archive, move the file `d8.exe` to the directory of your choice, and add the directory to the 'PATH' environment variable of the operating system.
3. Unlock the file `d8.exe`, for example, in the following way:
   - Right-click on the file and select *Properties* from the context menu.
   - In the Properties window, make sure you are on the *General* tab.
   - At the bottom of the General tab, you may see a section that says *Security* with a message like *This file came from another computer and might be blocked to help protect this computer.* If this message is displayed, there will be an *Unblock* button or checkbox next to it.
   - Check the *Unblock* checkbox or click the *Unblock* button, then click *Apply* and *OK* to save the changes.
4. Check that the CLI is working:

```
d8 help
```

5. Enable auto-completion in PowerShell by running the following command:

```
d8 completion powershell >> $PROFILE
```

Congrats, you have successfully installed Deckhouse CLI.

Deckhouse CLI command reference

> **i** This page is generated automatically.

d8 controls the Deckhouse Kubernetes Platform

Version: v0.32.5

Common options:

- `--help`, `-h`
  help for d8
- `--version`, `-v`
  version for d8

d8 backup

Backup various parts of Deckhouse Kubernetes Platform

Options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 backup cluster-config

Usage:

```
d8 backup cluster-config <backup-tarball-path> [options]
```

Take a snapshot of cluster configuration

Options

- `--compress`, `-c`
  Compress result file

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 backup etcd

Usage:

```
d8 backup etcd <snapshot-path> [options]
```

Take a snapshot of ETCD state

Options

- `--etcd-pod`, `-p`
  Name of the etcd pod to snapshot from. (optional)
- `--verbose`
  Verbose log output.

Global options

- `--context`

The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 backup loki

Usage:

```
d8 backup loki [options]
```

Dump logs from Loki api.

Options

- `--days`

  Limit maximum number of days in range to output per queue from Loki. (Default 5 maximum number of days in range per queue.)
- `--end`

  Set end timestamp range to dump logs from Loki. (Default get max end timestamp from Loki, ex. "2025-01-14 15:04:05".)
- `--limit`

  Limit the number of lines to output per queue from Loki. (Default 5000 max limit strings per queue.)
- `--start`

  Set start timestamp range to dump logs from Loki. (Default get max start timestamp from Loki, ex. "2025-01-12 15:04:05".)

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 cli

Manage the deckhouse-cli (d8) binary

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.
- `--rpp-insecure-skip-tls-verify`

  Skip registry-packages-proxy TLS verification. For debugging only.

d8 cli check

Usage:

```
d8 cli check [options]
```

Report whether a newer deckhouse-cli version is available

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.
- `--rpp-insecure-skip-tls-verify`

  Skip registry-packages-proxy TLS verification. For debugging only.

d8 cli update

Usage:

```
d8 cli update [options]
```

Update deckhouse-cli to the latest version

Options

- `--version`

  Exact version to install; downgrades are allowed (default: the latest).

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.
- `--rpp-insecure-skip-tls-verify`

  Skip registry-packages-proxy TLS verification. For debugging only.

d8 cli use

Usage:

```
d8 cli use <version> [options]
```

Switch to a specific deckhouse-cli version (no download when it is installed locally)

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.
- `--rpp-insecure-skip-tls-verify`

  Skip registry-packages-proxy TLS verification. For debugging only.

d8 cli versions

Alias: `list`

Usage:

```
d8 cli versions [options]
```

List deckhouse-cli versions available in the cluster registry

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--rpp-ca-file`

  Path to a PEM CA bundle used to verify the registry-packages-proxy TLS certificate. Defaults to $D8_RPP_CA_FILE.
- `--rpp-endpoint`

  registry-packages-proxy base URL (e.g. https://master:4219). Discovered from the cluster when empty. Defaults to $D8_RPP_ENDPOINT.
- `--rpp-insecure-skip-tls-verify`

  Skip registry-packages-proxy TLS verification. For debugging only.

d8 completion

Generate the autocompletion script for the specified shell

d8 completion bash

Usage:

```
d8 completion bash [options]
```

Generate the autocompletion script for bash

Options

- `--no-descriptions`

  disable completion descriptions

d8 completion fish

Usage:

```
d8 completion fish [options]
```

Generate the autocompletion script for fish

Options

- `--no-descriptions`

  disable completion descriptions

d8 completion powershell

Usage:

```
d8 completion powershell [options]
```

Generate the autocompletion script for powershell

Options

- `--no-descriptions`

  disable completion descriptions

d8 completion zsh

Usage:

```
d8 completion zsh [options]
```

Generate the autocompletion script for zsh

Options

- `--no-descriptions`

  disable completion descriptions

d8 cr

Work with container registries

Options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr catalog

Usage:

```
d8 cr catalog REGISTRY [options]
```

List the repositories in a registry

Options

- `--full-ref`

  Print the full repository reference (registry/repo)

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr config

Usage:

```
d8 cr config IMAGE [options]
```

Print the config of an image

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr digest

Usage:

```
d8 cr digest [IMAGE] [options]
```

Print the digest of an image

Options

- `--full-ref`

  Print the full image reference with digest (registry/repo@sha256:...); incompatible with --tarball
- `--tarball`

  Read the digest from a local tarball instead of the registry

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr export

Usage:

```
d8 cr export IMAGE [TARBALL] [options]
```

Export the filesystem of an image as a tarball

Global options

- `--allow-nondistributable-artifacts`
  Include non-distributable (foreign) layers when pushing
- `--insecure`
  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`
  Registry password or token (overrides the Docker config; use with --username)
- `--platform`
  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`
  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`
  Enable debug logs on stderr

d8 cr fs

Usage:

```
d8 cr fs [options]
```

Inspect or extract files inside a container image

Global options

- `--allow-nondistributable-artifacts`
  Include non-distributable (foreign) layers when pushing
- `--insecure`
  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`
  Registry password or token (overrides the Docker config; use with --username)
- `--platform`
  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`
  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`
  Enable debug logs on stderr

d8 cr fs cat

Usage:

```
d8 cr fs cat IMAGE PATH [options]
```

Print a file from a container image

Global options

- `--allow-nondistributable-artifacts`
  Include non-distributable (foreign) layers when pushing
- `--insecure`
  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`
  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr fs extract

Usage:

```
d8 cr fs extract IMAGE [options]
```

Extract a container image filesystem to a local directory

Options

- `--output`, `-o`

  Write the filesystem into this directory

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr fs ls

Usage:

```
d8 cr fs ls IMAGE [PATH] [options]
```

List files inside a container image

Options

- `--long`, `-l`

  Long format with mode and size

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr fs tree

Usage:

```
d8 cr fs tree IMAGE [PATH] [options]
```

Show the filesystem of a container image as a tree

Options

- `--depth`, `-L`

  Max depth to descend (0 = unlimited)
- `--dirsfirst`

  List directories before files
- `--size`

  Show file sizes (human-readable: B / KB / MB)

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr login

Usage:

```
d8 cr login [REGISTRY] [options]
```

Log in to a container registry

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr ls

Usage:

```
d8 cr ls REPO [options]
```

List the tags in a repository

Options

- `--full-ref`

  Print the full image reference (registry/repo:tag)
- `--omit-digest-tags`, `-0`

  Skip digest-based tags (sha256-*) created by signing tools

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr manifest

Usage:

```
d8 cr manifest IMAGE [options]
```

Print the manifest of an image

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr pull

Usage:

```
d8 cr pull IMAGE... PATH [options]
```

Pull one or more remote images to a local path

Options

- `--cache-path`, `-c`

  Cache image layers under this directory (reused between pulls)
- `--format`

  Output format (one of: tarball, legacy, oci)

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 cr push

Usage:

```
d8 cr push PATH IMAGE [options]
```

Push a local image to a registry

Options

- `--image-refs`

  Persist the pushed reference (with digest) to this file
- `--index`

  Push a multi-manifest OCI layout as an index (OCI layout dirs only)

Global options

- `--allow-nondistributable-artifacts`

  Include non-distributable (foreign) layers when pushing
- `--insecure`

  Allow plain HTTP and skip TLS verification (localhost and RFC1918 hosts already auto-allow HTTP)
- `--password`, `-p`

  Registry password or token (overrides the Docker config; use with --username)
- `--platform`

  Resolve images to platform os/arch[/variant][:osversion] (image-level commands only)
- `--username`, `-u`

  Registry username (overrides the Docker config; use with --password)
- `--verbose`, `-v`

  Enable debug logs on stderr

d8 data

Data operations (export/import)

d8 data create

Usage:

```
d8 data create [flags] data_export_name volume_type/volume_name [options]
```

Deprecated: use 'd8 data export create'

Options

- `--namespace`, `-n`

  data volume namespace

- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live

d8 data delete

Usage:

```
d8 data delete [flags] data_export_name [options]
```

Deprecated: use 'd8 data export delete'

Options

- `--namespace`, `-n`

  data volume namespace

d8 data download

Usage:

```
d8 data download [flags] [KIND/]data_export_name [path/file.ext] [options]
```

Deprecated: use 'd8 data export download'

Options

- `--namespace`, `-n`

  data volume namespace
- `--output`, `-o`

  file to save data (default: same as resource)
- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live for auto-created DataExport

d8 data export

Usage:

```
d8 data export [options]
```

Export data (DataExport)

d8 data export create

Usage:

```
d8 data export create [flags] data_export_name volume_type/volume_name [options]
```

Create dataexport kubernetes resource

Options

- `--namespace`, `-n`

  data volume namespace
- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live

d8 data export delete

Usage:

```
d8 data export delete [flags] data_export_name [options]
```

Delete dataexport kubernetes resource

Options

- `--namespace`, `-n`

  data volume namespace

d8 data export download

Usage:

```
d8 data export download [flags] [KIND/]data_export_name [path/file.ext] [options]
```

Download exported data

Options

- `--cleanup`

  Delete auto-created DataExport without prompting (--cleanup=true to delete, --cleanup=false to keep)
- `--namespace`, `-n`

  data volume namespace
- `--output`, `-o`

  file to save data (default: same as resource)
- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live for auto-created DataExport

d8 data export list

Alias: `ls`

Usage:

```
d8 data export list [flags] data_export_name [/path/] [options]
```

List DataExported content information

Options

- `--cleanup`

  Delete auto-created DataExport without prompting (--cleanup=true to delete, --cleanup=false to keep)
- `--namespace`, `-n`

  data volume namespace
- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live for auto-created DataExport

d8 data import

Usage:

```
d8 data import [options]
```

Import data (DataImport)

d8 data import create

Usage:

```
d8 data import create [flags] data_import_name [options]
```

Create DataImport

Options

- `--file`, `-f`
  PVC manifest file path
- `--namespace`, `-n`
  data volume namespace
- `--publish`
  Provide access outside of cluster
- `--ttl`
  Time to live
- `--wffc`
  Wait for first consumer

d8 data import delete

Usage:

```
d8 data import delete [flags] data_import_name [options]
```

Delete dataimport kubernetes resource

Options

- `--namespace`, `-n`
  data volume namespace

d8 data import upload

Usage:

```
d8 data import upload [flags] data_import_name path/file.ext [options]
```

Upload a file to the provided url

Options

- `--chunks`, `-c`
  number of chunks to upload
- `--dstPath`, `-d`
  destination path of the uploaded file
- `--file`, `-f`
  file to upload
- `--namespace`, `-n`
  data volume namespace
- `--publish`, `-P`
  publish the uploaded file
- `--resume`
  resume upload if process was interrupted

d8 data list

Usage:

```
d8 data list [flags] data_export_name [/path/] [options]
```

Deprecated: use 'd8 data export list'

Options

- `--namespace`, `-n`

  data volume namespace
- `--publish`

  Provide access outside of cluster
- `--ttl`

  Time to live for auto-created DataExport

d8 delivery-kit

Alias: `dk`

A set of tools for building, distributing, and deploying containerized applications

d8 delivery-kit build

Usage:

```
d8 delivery-kit build [IMAGE_NAME...] [options]
```

Build images

Options

- `--add-custom-tag`

  Set tag alias for the content-based tag.
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also can be defined with $WERF_ADD_CUSTOM_TAG_* (e.g. $WERF_ADD_CUSTOM_TAG_1="%image%-tag1", $WERF_ADD_CUSTOM_TAG_2="%image%-tag2")
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache

(default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)

- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--check-built-images`

  Check that all used images are previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_CHECK_BUILT_IMAGES)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go,

$WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree

- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Check that all used images are previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)

- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)

d8 delivery-kit bundle

Usage:

```
d8 delivery-kit bundle [options]
```

Work with werf bundles: publish bundles into container registry and deploy bundles into Kubernetes cluster

d8 delivery-kit bundle apply

Usage:

```
d8 delivery-kit bundle apply [options]
```

Apply bundle into Kubernetes

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--atomic`

  Enable auto rollback of the failed release to the previous deployed release version when current deploy process have failed ($WERF_ATOMIC by default)
- `--auto-rollback`, `-R`

  Enable auto rollback of the failed release to the previous deployed release version when current deploy process have failed ($WERF_AUTO_ROLLBACK by default)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--delete-propagation`

  Set default delete propagation strategy (default $WERF_DELETE_PROPAGATION or Foreground).
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--deploy-graph-path`

  Save deploy graph path to the specified file (by default $WERF_DEPLOY_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--deploy-report-path`

  Change deploy report path and format (by default $WERF_DEPLOY_REPORT_PATH or ".werf-deploy-report.json" if not set). Extension must be .json for JSON format. If extension not specified, then .json is used
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--env`

  Use specified environment (default $WERF_ENV)

- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with

$WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`

  Use specified Kubernetes namespace (default $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)

- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-install-crds`

  Do not install CRDs from "crds/" directories of installed charts (default $WERF_NO_INSTALL_CRDS)
- `--no-notes`

  Don't show release notes at the end of the release (default $WERF_NO_NOTES)
- `--no-pod-logs`

  Disable Pod logs collection and printing (default $WERF_NO_POD_LOGS or false)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default $WERF_RELEASE)
- `--release-info-annotations`

  Add annotations to release metadata (default $WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`

  Add Helm release labels (can specify multiple). Kind of labels depends or release storage driver. Format: labelName=labelValue.
  Also, can be specified with $WERF_RELEASE_LABEL_* (e.g. $WERF_RELEASE_LABEL_1=labelName1=labelValue1, $WERF_RELEASE_LABEL_2=labelName2=labelValue2)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--render-subchart-notes`

  If set, render subchart notes along with the parent (by default $WERF_RENDER_SUBCHART_NOTES or false)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)

- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--rollback-graph-path`

  Save rollback graph path to the specified file (by default $WERF_ROLLBACK_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--runtime-annotations`

  Add annotations which will not trigger resource updates to all resources (default $WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`

  Add labels which will not trigger resource updates to all resources (default $WERF_RUNTIME_LABELS)
- `--save-deploy-report`

  Save deploy report (by default $WERF_SAVE_DEPLOY_REPORT or false). Its path and format configured with --deploy-report-path
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead,

unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)

- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--tag`

  Provide exact tag version or semver-based pattern, werf will install or upgrade to the latest version of the specified bundle ($WERF_TAG or latest by default)
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--timeout`, `-t`

  Resources tracking timeout in seconds ($WERF_TIMEOUT by default)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit bundle copy

Usage:

```
d8 delivery-kit bundle copy [options]
```

Copy published bundle into another location

Options

- `--container-registry-mirror`

(Buildah-only) Use specified mirrors for docker.io

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repos
- `--from`

  Source address of the bundle to copy, specify bundle archive using schema `archive:PATH_TO_ARCHIVE.tar.gz`, specify remote bundle with schema `[docker://]REPO:TAG` or without schema.
- `--helm-compatible-chart`, `-C`

  Set chart name in the Chart.yaml of the published chart to the last path component of container registry repo (for REGISTRY/PATH/TO/REPO address chart name will be REPO, more info https://helm.sh/docs/topics/registries/#oci-feature-deprecation-and-behavior-changes-with-v370). In helm compatibility mode chart is fully conforming with the helm OCI registry requirements. Default true or $WERF_HELM_COMPATIBLE_CHART.
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--rename-chart`

  Force setting of chart name in the Chart.yaml of the published chart to the specified value (can be

set by the $WERF_RENAME_CHART, no rename by default, could not be used together with the '--helm-compatible-chart' option).

- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--to`

  Destination address of the bundle to copy, specify bundle archive using schema `archive:PATH_TO_ARCHIVE.tar.gz`, specify remote bundle with schema `[docker://]REPO:TAG` or without schema.
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit bundle plan

Usage:

```
d8 delivery-kit bundle plan [options]
```

Plan bundle apply into Kubernetes

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--delete-propagation`

  Set default delete propagation strategy (default $WERF_DELETE_PROPAGATION or Foreground).
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--deploy-graph-path`

  Save deploy graph path to the specified file (by default $WERF_DEPLOY_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--diff-context-lines`

  Show N lines of context around diffs ($WERF_DIFF_CONTEXT_LINES by default)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default

$WERF_DISABLE_DEFAULT_VALUES or false)

- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--env`

  Use specified environment (default $WERF_ENV)
- `--exit-code`

  Return exit code 0 if no changes, 1 if error, 2 if any changes planned (default $WERF_EXIT_CODE or false)
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or

$WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).

- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`

  Use specified Kubernetes namespace (default $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-install-crds`

  Do not install CRDs from "crds/" directories of installed charts (default $WERF_NO_INSTALL_CRDS)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default $WERF_RELEASE)
- `--release-info-annotations`

  Add annotations to release metadata (default $WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`

  Add Helm release labels (can specify multiple). Kind of labels depends or release storage driver. Format: labelName=labelValue.
  Also, can be specified with $WERF_RELEASE_LABEL_* (e.g. $WERF_RELEASE_LABEL_1=labelName1=labelValue1, $WERF_RELEASE_LABEL_2=labelName2=labelValue2)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)

- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--runtime-annotations`

  Add annotations which will not trigger resource updates to all resources (default $WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`

  Add labels which will not trigger resource updates to all resources (default $WERF_RUNTIME_LABELS)
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing.

Can specify multiple or separate values with commas: key1=val1,key2=val2.
Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)

- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--show-insignificant-diffs`

  Show insignificant diff lines ($WERF_SHOW_INSIGNIFICANT_DIFFS by default)
- `--show-sensitive-diffs`

  Show sensitive diff lines ($WERF_SHOW_SENSITIVE_DIFFS by default)
- `--show-verbose-crd-diffs`

  Show verbose CRD diff lines ($WERF_SHOW_VERBOSE_CRD_DIFFS by default)
- `--show-verbose-diffs`

  Show verbose diff lines ($WERF_SHOW_VERBOSE_DIFFS by default)
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tag`

  Provide exact tag version or semver-based pattern, werf will install or upgrade to the latest version of the specified bundle ($WERF_TAG or latest by default)
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit bundle publish

Usage:

```
d8 delivery-kit bundle publish [IMAGE_NAME...] [options]
```

Publish bundle

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1,

$WERF_ADD_ANNOTATION_2=annoName2=annoValue2)

- `--add-custom-tag`

  Set tag alias for the content-based tag.
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also can be defined with $WERF_ADD_CUSTOM_TAG_* (e.g. $WERF_ADD_CUSTOM_TAG_1="%image%-tag1", $WERF_ADD_CUSTOM_TAG_2="%image%-tag2")
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option

- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)

Command needs granted permissions to read, pull and push images into the specified repo and to pull base images

- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--helm-compatible-chart`, `-C`

  Set chart name in the Chart.yaml of the published chart to the last path component of container registry repo (for REGISTRY/PATH/TO/REPO address chart name will be REPO, more info https://helm.sh/docs/topics/registries/#oci-feature-deprecation-and-behavior-changes-with-v370). In helm compatibility mode chart is fully conforming with the helm OCI registry requirements. Default false or $WERF_HELM_COMPATIBLE_CHART.
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--ignore-secret-key`

Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)

- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

Enable verbose output (default $WERF_LOG_VERBOSE).

- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--rename-chart`

  Force setting of chart name in the Chart.yaml of the published chart to the specified value (can be set by the $WERF_RENAME_CHART, no rename by default, could not be used together with the '--helm-compatible-chart' option).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)

- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)

- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tag`

  Publish bundle into container registry repo by the provided tag ($WERF_TAG or latest by default)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--use-custom-tag`

  Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also, can be defined with $WERF_USE_CUSTOM_TAG (e.g. $WERF_USE_CUSTOM_TAG="%image%-tag")
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state

($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit bundle render

Usage:

```
d8 delivery-kit bundle render [options]
```

Render Kubernetes manifests from bundle

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--bundle-dir`, `-b`

  Get extracted bundle from directory instead of registry (default $WERF_BUNDLE_DIR)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--env`

  Use specified environment (default $WERF_ENV)
- `--extra-apiversions`

  Extra Kubernetes API versions passed to $.Capabilities.APIVersions. Can be also set with $WERF_EXTRA_APIVERSIONS_* environment variables, values can be comma-separated
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default

$WERF_FORCE_ADOPTION or false)

- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--include-crds`

  Include CRDs in the templated output (default $WERF_INCLUDE_CRDS)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)

- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--kube-version`

  Set specific Capabilities.KubeVersion (default $WERF_KUBE_VERSION)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`

  Use specified Kubernetes namespace (default $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)

- `--output`
  Write render output to the specified file instead of stdout ($WERF_RENDER_OUTPUT by default)
- `--provenance-keyring`
  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`
  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`
  Disable explanatory output (default $WERF_QUIET).
- `--release`
  Use specified Helm release name (default $WERF_RELEASE)
- `--release-storage`
  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`
  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--repo`
  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`
  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--secret-key`
  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`
  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`
  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`
  Shortcut to set current docker config into the .Values.dockerconfigjson
- `--set-file`
  Set values from respective files specified via the command line (can specify multiple or separate

values with commas: key1=path1,key2=path2).
Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)

- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--show-only`, `-s`

  only show manifests rendered from the given templates
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tag`

  Provide exact tag version or semver-based pattern, werf will render the latest version of the specified bundle ($WERF_TAG or latest by default)
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--validate`

  Validate your manifests against the Kubernetes cluster you are currently pointing at (default

$WERF_VALIDATE)
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit chart

Usage:

```
d8 delivery-kit chart [options]
```

Work with charts

d8 delivery-kit chart ts

Usage:

```
d8 delivery-kit chart ts [options]
```

Work with TypeScript charts

d8 delivery-kit chart ts build

Usage:

```
d8 delivery-kit chart ts build [PATH] [options]
```

Build TypeScript chart.

Options

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

Use specified environment (default $WERF_ENV)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit chart ts init

Usage:

```
d8 delivery-kit chart ts init [PATH] [options]
```

Initialize the files needed to render manifests using TypeScript.

Options

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit ci-env

Usage:

```
d8 delivery-kit ci-env CI_SYSTEM [options]
```

Generate werf environment variables for specified CI system

Options

- `--allow-includes-update`
  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--as-env-file`
  Create the .env file and print the path for sourcing (default $WERF_AS_ENV_FILE).
- `--as-file`
  Create the script and print the path for sourcing (default $WERF_AS_FILE).
- `--config`
  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`
  Custom path for storing rendered configuration file
- `--config-templates-dir`
  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--debug-templates`
  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`
  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`
  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command will copy specified or default (~/.docker) config to the temporary directory and may perform additional login with new config.
- `--env`
  Use specified environment (default $WERF_ENV)
- `--git-work-tree`
  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)

- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--login-to-registry`

  Log in to CI-specific registry automatically if possible (default $WERF_LOGIN_TO_REGISTRY).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to custom file (default $WERF_OUTPUT_FILE_PATH).
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--shell`

  Set to cmdexe, powershell or use the default behavior that is compatible with any unix shell (default $WERF_SHELL).
- `--tagging-strategy`

  stub
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-docker-auth-config`

  Generate Docker config from DOCKER_AUTH_CONFIG environment variable instead of copying current Docker config (default $WERF_USE_DOCKER_AUTH_CONFIG).
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit cleanup

Usage:

```
d8 delivery-kit cleanup [options]
```

Cleanup project images in the container registry

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

Enable development mode (default $WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and delete images from the specified repo
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)

- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--keep-list`

  Set path to keep list file (default $WERF_KEEP_LIST)
- `--keep-stages-built-within-last-n-hours`

  Keep stages that were built within last hours (default $WERF_KEEP_STAGES_BUILT_WITHIN_LAST_N_HOURS or 2)
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Scan for used images only in the specified kube context, scan all contexts from kube config otherwise (default false or $WERF_SCAN_CONTEXT_ONLY)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

Enable verbose output (default $WERF_LOG_VERBOSE).

- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--scan-context-namespace-only`

  Scan for used images only in namespace linked with context for each available context in kube-config (or only for the context specified with option --kube-context). When disabled will scan all namespaces in all contexts (or only for the context specified with option --kube-context). (Default $WERF_SCAN_CONTEXT_NAMESPACE_ONLY)
- `--scan-context-only`

  Scan for used images only in the specified kube context, scan all contexts from kube config otherwise (default false or $WERF_SCAN_CONTEXT_ONLY)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default

$WERF_SKIP_TLS_VERIFY_REGISTRY)

- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--without-kube`

  Do not skip deployed Kubernetes images (default $WERF_WITHOUT_KUBE)

d8 delivery-kit completion

Usage:

```
d8 delivery-kit completion [options]
```

Generate bash completion scripts

Options

- `--shell`

  Set to bash, zsh, fish or powershell (default $WERF_SHELL or bash)

d8 delivery-kit compose

Usage:

```
d8 delivery-kit compose [options]
```

Work with docker-compose

d8 delivery-kit compose config

Usage:

```
d8 delivery-kit compose config [IMAGE_NAME...] [options] [--docker-compose-options="OPTIONS"] [--docker-
compose-command-options="OPTIONS"]
```

Run docker-compose config command with forwarded image names.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)

- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)

- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-compose-bin-path`

  DEPRECATED: "docker compose" command always used now, this option is ignored. Define docker-compose bin path (default $WERF_DOCKER_COMPOSE_BIN_PATH)
- `--docker-compose-command-options`

  Define docker-compose command options (default $WERF_DOCKER_COMPOSE_COMMAND_OPTIONS)
- `--docker-compose-options`

  Define docker-compose options (default $WERF_DOCKER_COMPOSE_OPTIONS)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).

- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default

$WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit compose down

Usage:

```
d8 delivery-kit compose down [IMAGE_NAME...] [options] [--docker-compose-options="OPTIONS"] [--docker-compose-command-options="OPTIONS"]
```

Run docker-compose down command with forwarded image names.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)

- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)

- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-compose-bin-path`

  DEPRECATED: "docker compose" command always used now, this option is ignored. Define docker-compose bin path (default $WERF_DOCKER_COMPOSE_BIN_PATH)
- `--docker-compose-command-options`

  Define docker-compose command options (default $WERF_DOCKER_COMPOSE_COMMAND_OPTIONS)
- `--docker-compose-options`

  Define docker-compose options (default $WERF_DOCKER_COMPOSE_OPTIONS)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

Container registry storage address (default $WERF_FINAL_REPO)

- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).

- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).

When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option

- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit compose run

Usage:

```
d8 delivery-kit compose run [IMAGE_NAME...] [options] [--docker-compose-options="OPTIONS"] [--docker-compose-command-options="OPTIONS"] -- SERVICE [COMMAND] [ARGS...]
```

Run docker-compose run command with forwarded image names.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing

images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-compose-bin-path`

  DEPRECATED: "docker compose" command always used now, this option is ignored. Define docker-compose bin path (default $WERF_DOCKER_COMPOSE_BIN_PATH)
- `--docker-compose-command-options`

  Define docker-compose command options (default $WERF_DOCKER_COMPOSE_COMMAND_OPTIONS)
- `--docker-compose-options`

  Define docker-compose options (default $WERF_DOCKER_COMPOSE_OPTIONS)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)

- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit compose up

Usage:

```
d8 delivery-kit compose up [IMAGE_NAME...] [options] [--docker-compose-options="OPTIONS"] [--docker-compose-command-options="OPTIONS"] [--] [SERVICE...]
```

Run docker-compose up command with forwarded image names.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used

- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-compose-bin-path`

  DEPRECATED: "docker compose" command always used now, this option is ignored. Define docker-compose bin path (default $WERF_DOCKER_COMPOSE_BIN_PATH)
- `--docker-compose-command-options`

  Define docker-compose command options (default $WERF_DOCKER_COMPOSE_COMMAND_OPTIONS)
- `--docker-compose-options`

  Define docker-compose options (default $WERF_DOCKER_COMPOSE_OPTIONS)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage

volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)

- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)

- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)

- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit config

Usage:

```
d8 delivery-kit config [options]
```

Work with werf.yaml

d8 delivery-kit config graph

Usage:

```
d8 delivery-kit config graph [IMAGE_NAME...] [options]
```

Print dependency graph for images in werf.yaml.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit config list

Usage:

```
d8 delivery-kit config list [options]
```

List image and artifact names defined in werf.yaml.

Options

- `--allow-includes-update`

Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Show only final images
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--images-only`

  Show only final images
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit config render

Usage:

```
d8 delivery-kit config render [IMAGE_NAME...] [options]
```

Render werf.yaml.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains

.git in the current or parent directories)

- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit converge

Usage:

```
d8 delivery-kit converge [options]
```

Build and push images, then deploy application into Kubernetes

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g.
  $WERF_ADD_ANNOTATION_1=annoName1=annoValue1,
  $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-custom-tag`

  Set tag alias for the content-based tag.
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is

more than one image in the werf config);
- %image_content_based_tag% to use a content-based tag.
For cleaning custom tags and associated content-based tag are treated as one.
Also can be defined with $WERF_ADD_CUSTOM_TAG_* (e.g. $WERF_ADD_CUSTOM_TAG_1="%image%-tag1", $WERF_ADD_CUSTOM_TAG_2="%image%-tag2")

- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--atomic`

  Enable auto rollback of the failed release to the previous deployed release version when current deploy process have failed ($WERF_ATOMIC by default)
- `--auto-rollback`, `-R`

  Enable auto rollback of the failed release to the previous deployed release version when current deploy process have failed ($WERF_AUTO_ROLLBACK by default)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)

- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--delete-propagation`

  Set default delete propagation strategy (default $WERF_DELETE_PROPAGATION or Foreground).
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--deploy-graph-path`

  Save deploy graph path to the specified file (by default $WERF_DEPLOY_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--deploy-report-path`

  Change deploy report path and format (by default $WERF_DEPLOY_REPORT_PATH or ".werf-deploy-report.json" if not set). Extension must be .json for JSON format. If extension not specified, then .json is used
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)

- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked

files) in the git repository worktree
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)

- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-install-crds`

  Do not install CRDs from "crds/" directories of installed charts (default $WERF_NO_INSTALL_CRDS)
- `--no-notes`

  Don't show release notes at the end of the release (default $WERF_NO_NOTES)
- `--no-pod-logs`

  Disable Pod logs collection and printing (default $WERF_NO_POD_LOGS or false)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--plan-lifetime`

  How long plan artifact is valid
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default

$WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or $WERF_RELEASE)
- `--release-info-annotations`

  Add annotations to release metadata (default $WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`

  Add Helm release labels (can specify multiple). Kind of labels depends or release storage driver. Format: labelName=labelValue.
  Also, can be specified with $WERF_RELEASE_LABEL_* (e.g. $WERF_RELEASE_LABEL_1=labelName1=labelValue1, $WERF_RELEASE_LABEL_2=labelName2=labelValue2)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--render-subchart-notes`

  If set, render subchart notes along with the parent (by default $WERF_RENDER_SUBCHART_NOTES or false)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--rollback-graph-path`

  Save rollback graph path to the specified file (by default $WERF_ROLLBACK_GRAPH_PATH).

Extension must be .dot or not specified. If extension not specified, then .dot is used

- `--runtime-annotations`

  Add annotations which will not trigger resource updates to all resources (default $WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`

  Add labels which will not trigger resource updates to all resources (default $WERF_RUNTIME_LABELS)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--save-deploy-report`

  Save deploy report (by default $WERF_SAVE_DEPLOY_REPORT or false). Its path and format configured with --deploy-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)

- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or

- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--timeout`, `-t`

  Resources tracking timeout in seconds ($WERF_TIMEOUT by default)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--use-custom-tag`

  Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also, can be defined with $WERF_USE_CUSTOM_TAG (e.g. $WERF_USE_CUSTOM_TAG="%image%-tag")
- `--use-plan`

  Use the gzip-compressed JSON plan file from the specified path during release install
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit cr

Usage:

```
d8 delivery-kit cr [options]
```

Authenticate with container registry.

d8 delivery-kit cr login

Usage:

```
d8 delivery-kit cr login registry [options]
```

Login into remote registry.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).

- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--password`, `-p`

  Use specified password for login (default $WERF_PASSWORD)
- `--password-stdin`

  Read password from stdin for login (default $WERF_PASSWORD_STDIN)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--username`, `-u`

  Use specified username for login (default $WERF_USERNAME)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit cr logout

Usage:

```
d8 delivery-kit cr logout registry [options]
```

Logout from a remote registry.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit dismiss

Usage:

```
d8 delivery-kit dismiss [options]
```

Delete werf release from Kubernetes

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)

- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--deploy-report-path`

  Change deploy report path and format (by default $WERF_DEPLOY_REPORT_PATH or ".werf-deploy-report.json" if not set). Extension must be .json for JSON format. If extension not specified, then .json is used
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage

volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)

- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)

- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-pod-logs`

  Disable Pod logs collection and printing (default $WERF_NO_POD_LOGS or false)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or $WERF_RELEASE)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

Choose repo container registry implementation.
The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--timeout`, `-t`

  Resources tracking timeout in seconds ($WERF_TIMEOUT by default)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-deploy-report`

  Use deploy report, previously saved with --save-deploy-report (by default $WERF_USE_DEPLOY_REPORT or false). Its path and format configured with --deploy-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--with-hooks`

  Delete Helm Release hooks getting from existing revisions (default $WERF_WITH_HOOKS or true)
- `--with-namespace`

Delete Kubernetes Namespace after purging Helm Release (default $WERF_WITH_NAMESPACE)

d8 delivery-kit export

Usage:

```
d8 delivery-kit export [IMAGE_NAME...] [options]
```

Export images

Options

- `--add-label`

  Add label to exported images (can specify multiple).
  Format: labelName=labelValue[labelName=labelValue ...]. The default separator is a newline ("\n"), but it can be customized using the --add-label-separator flag.
  Also, can be specified with $WERF_EXPORT_ADD_LABEL_* (e.g. $WERF_EXPORT_ADD_LABEL_1=labelName1=labelValue1, $WERF_EXPORT_ADD_LABEL_2=labelName2=labelValue2)
- `--add-label-separator`

  Separator for --add-label values (default $WERF_EXPORT_ADD_LABEL_SEPARATOR or "\n")
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)

- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)

- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)

- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tag`

  Set a tag template (can specify multiple).
  It is necessary to use image name shortcut %image% or %image_slug% if multiple images are exported (e.g. REPO:TAG-%image% or REPO-%image%:TAG)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)

d8 delivery-kit helm

Usage:

```
d8 delivery-kit helm [options]
```

Manage application deployment with helm

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or

~/.docker (in the order of priority)

- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm create

Usage:

```
d8 delivery-kit helm create NAME [options]
```

create a new chart with the given name

Options

- `--starter`, `-p`

  the name or absolute path to Helm starter scaffold

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm dependency

Aliases: `dep`, `dependencies`

Usage:

```
d8 delivery-kit helm dependency update|build|list [options]
```

manage a chart's dependencies

Global options

- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`
  No-op
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`
  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`
  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`
  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`
  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`
  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`
  namespace scope for this request
- `--quiet`
  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`
  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm dependency build

Usage:

```
d8 delivery-kit helm dependency update|build|list build CHART [options]
```

rebuild the charts/ directory based on the Chart.lock file

Options

- `--keyring`

  keyring containing public keys
- `--skip-refresh`

  do not refresh the local repository cache
- `--verify`

  verify the packages against signatures

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm dependency list

Alias: `ls`

Usage:

```
d8 delivery-kit helm dependency update|build|list list CHART [options]
```

list the dependencies for the given chart

Options

- `--max-col-width`

  maximum column width for output table

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm dependency update

Alias: `up`

Usage:

```
d8 delivery-kit helm dependency update|build|list update CHART [options]
```

update charts/ based on the contents of Chart.yaml

Options

- `--keyring`

  keyring containing public keys
- `--skip-refresh`

  do not refresh the local repository cache
- `--verify`

  verify the packages against signatures

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

Enable verbose output (default $WERF_LOG_VERBOSE).

- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm env

Usage:

```
d8 delivery-kit helm env [options]
```

helm client environment information

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request

- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get

Usage:

```
d8 delivery-kit helm get [options]
```

download extended information of a named release

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get all

Usage:

```
d8 delivery-kit helm get all RELEASE_NAME [options]
```

download all information for a named release

Options

- `--revision`

  get the named release with revision
- `--template`

  go template for formatting the output, eg: {{.Release.Name}}

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request

- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get hooks

Usage:

```
d8 delivery-kit helm get hooks RELEASE_NAME [options]
```

download all hooks for a named release

Options

- `--revision`

  get the named release with revision

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get manifest

Usage:

```
d8 delivery-kit helm get manifest RELEASE_NAME [options]
```

download the manifest for a named release

Options

- `--revision`

  get the named release with revision

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).

- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get metadata

Usage:

```
d8 delivery-kit helm get metadata RELEASE_NAME [options]
```

This command fetches metadata for a given release

Options

- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--revision`

  specify release revision

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get notes

Usage:

```
d8 delivery-kit helm get notes RELEASE_NAME [options]
```

download the notes for a named release

Options

- `--revision`

  get the named release with revision

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).

- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get values

Usage:

```
d8 delivery-kit helm get values RELEASE_NAME [options]
```

download the values file for a named release

Options

- `--all`, `-a`

  dump all (computed) values
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--revision`

  get the named release with revision

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.

Defaults to:
* $WERF_LOG_TERMINAL_WIDTH
* interactive terminal width or 140

- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm get-autogenerated-values

Usage:

```
d8 delivery-kit helm get-autogenerated-values [IMAGE_NAME...] [options]
```

Get service values yaml generated by werf for helm chart during deploy

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)

- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo and to pull base images
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)

- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)

- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.dockerconfigjson
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

Default:
- $WERF_SYNCHRONIZATION, or
- :local if --repo is not specified, or
- https://delivery-sync.deckhouse.ru if --repo has been specified.

The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only

- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--use-custom-tag`

  Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also, can be defined with $WERF_USE_CUSTOM_TAG (e.g. $WERF_USE_CUSTOM_TAG="%image%-tag")
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm get-namespace

Usage:

```
d8 delivery-kit helm get-namespace [options]
```

Print Kubernetes Namespace that will be used in current configuration with specified params.

Options

- `--allow-includes-update`

Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140

- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm get-release

Usage:

```
d8 delivery-kit helm get-release [options]
```

Print Helm Release name that will be used in current configuration with specified params.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

Enable development mode (default $WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).

- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm history

Alias: `hist`

Usage:

```
d8 delivery-kit helm history RELEASE_NAME [options]
```

fetch release history

Options

- `--max`

  maximum number of revision to include in history
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).

- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm install

Usage:

```
d8 delivery-kit helm install [NAME] [CHART] [options]
```

install a chart

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--atomic`

  if set, the installation process deletes the installation on failure. The --wait flag will be set automatically if --atomic is used
- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--cleanup-on-fail`

  allow deletion of new resources created in this installation when install fails
- `--create-namespace`

create the release namespace if not present
- `--dependency-update`

  update dependencies if they are missing before installing the chart
- `--deploy-report-path`

  save deploy report in JSON to the specified path
- `--description`

  add a custom description
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-openapi-validation`

  if set, the installation process will not validate rendered templates against the Kubernetes OpenAPI Schema
- `--dry-run`

  simulate an install. If --dry-run is set with no option being specified or as '--dry-run=client', it will not attempt cluster connections. Setting '--dry-run=server' allows attempting cluster connections.
- `--enable-dns`

  enable DNS lookups when rendering templates
- `--force`

  force resource updates through a replacement strategy
- `--generate-name`, `-g`

  generate the name (and omit the NAME parameter)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--labels`, `-l`

  Labels that would be added to release metadata. Should be divided by comma.
- `--name-template`

  specify template used to name the release
- `--no-hooks`

  prevent hooks from running during install
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--post-renderer`

  the path to an executable to be used for post rendering. If it exists in $PATH, the binary will be used, otherwise it will try to look for the executable at the given path
- `--post-renderer-args`

  an argument to the post-renderer (can specify multiple)
- `--render-subchart-notes`

  if set, render subchart notes along with the parent

- `--replace`

  re-use the given name, only if that name is a deleted release which remains in the history. This is unsafe in production
- `--repo`

  chart repository url where to locate the requested chart
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  set values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--set-file`

  set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2)
- `--set-json`

  set JSON values on the command line (can specify multiple or separate values with commas: key1=jsonval1,key2=jsonval2)
- `--set-literal`

  set a literal STRING value on the command line
- `--set-string`

  set STRING values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--skip-crds`

  if set, no CRDs will be installed. By default, CRDs are installed if not already present
- `--timeout`

  time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--username`

  chart repository username where to locate the requested chart
- `--values`, `-f`

  specify values in a YAML file or a URL (can specify multiple)
- `--verify`

  verify the package before using it
- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used
- `--wait`

  if set, will wait until all Pods, PVCs, Services, and minimum number of Pods of a Deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as --timeout
- `--wait-for-jobs`

  if set and --wait enabled, will wait until all Jobs have been completed before marking the release as successful. It will wait for as long as --timeout

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)

- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm lint

Usage:

```
d8 delivery-kit helm lint PATH [options]
```

examine a chart for possible issues

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).

Format: labelName=labelValue.
Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)

- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--kube-version`

  Kubernetes version used for capabilities and deprecation checks
- `--quiet`

  print only warnings and errors
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  set values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--set-file`

  set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2)
- `--set-json`

  set JSON values on the command line (can specify multiple or separate values with commas: key1=jsonval1,key2=jsonval2)
- `--set-literal`

  set a literal STRING value on the command line
- `--set-string`

  set STRING values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--strict`

  fail on lint warnings
- `--values`, `-f`

  specify values in a YAML file or a URL (can specify multiple)
- `--with-subcharts`

  lint dependent charts

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

Kubernetes config context (default $WERF_KUBE_CONTEXT)

- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm list

Alias: `ls`

Usage:

```
d8 delivery-kit helm list [options]
```

list releases

Options

- `--all`, `-a`

  show all releases without any filter applied
- `--all-namespaces`, `-A`

  list releases across all namespaces
- `--date`, `-d`

  sort by release date
- `--deployed`

  show deployed releases. If no other is specified, this will be automatically enabled
- `--failed`

  show failed releases
- `--filter`, `-f`

  a regular expression (Perl compatible). Any releases that match the expression will be included in the results
- `--max`, `-m`

  maximum number of releases to fetch

- `--no-headers`

  don't print headers when using the default output format
- `--offset`

  next release index in the list, used to offset from start value
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--pending`

  show pending releases
- `--reverse`, `-r`

  reverse the sort order
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', and '!='.(e.g. -l key1=value1,key2=value2). Works only for secret(default) and configmap storage backends.
- `--short`, `-q`

  output short (quiet) listing format
- `--superseded`

  show superseded releases
- `--time-format`

  format time using golang time formatter. Example: --time-format "2006-01-02 15:04:05Z0700"
- `--uninstalled`

  show uninstalled releases (if 'helm uninstall --keep-history' was used)
- `--uninstalling`

  show releases that are currently being uninstalled

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm package

Usage:

```
d8 delivery-kit helm package [CHART_PATH] [...] [options]
```

package a chart directory into a chart archive

Options

- `--app-version`

  set the appVersion on the chart to this version
- `--dependency-update`, `-u`

  update dependencies from "Chart.yaml" to dir "charts/" before packaging
- `--destination`, `-d`

  location to write the chart.
- `--key`

  name of the key to use when signing. Used if --sign is true
- `--keyring`

  location of a public keyring
- `--passphrase-file`

  location of a file which contains the passphrase for the signing key. Use "-" in order to read from stdin.
- `--sign`

  use a PGP private key to sign this package
- `--version`

  set the version on the chart to this semver version

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)

- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm plugin

Usage:

```
d8 delivery-kit helm plugin [options]
```

install, list, or uninstall Helm plugins

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm plugin install

Alias: `add`

Usage:

```
d8 delivery-kit helm plugin install [options] <path|url>...
```

install one or more Helm plugins

Options

- `--version`

  specify a version constraint. If this is not specified, the latest version is installed

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

Set log color mode.
Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`
  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`
  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`
  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`
  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`
  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`
  namespace scope for this request
- `--quiet`
  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`
  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm plugin list

Alias: `ls`

Usage:

```
d8 delivery-kit helm plugin list [options]
```

list installed Helm plugins

Global options

- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`
  No-op
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.

Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm plugin uninstall

Aliases: `rm`, `remove`

Usage:

```
d8 delivery-kit helm plugin uninstall <plugin>... [options]
```

uninstall one or more Helm plugins

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

Enable debug (default $WERF_LOG_DEBUG).

- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm plugin update

Alias: `up`

Usage:

```
d8 delivery-kit helm plugin update <plugin>... [options]
```

update one or more Helm plugins

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).

- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.

  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH

  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm pull

Alias: `fetch`

Usage:

```
d8 delivery-kit helm pull [chart URL | repo/chartname] [...] [options]
```

download a chart from a repository and (optionally) unpack it in local directory

Options

- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--destination`, `-d`

  location to write the chart. If this and untardir are specified, untardir is appended to this
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored.
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--prov`

fetch the provenance file, but don't perform verification
- `--repo`

  chart repository url where to locate the requested chart
- `--untar`

  if set to true, will untar the chart after downloading it
- `--untardir`

  if untar is specified, this flag specifies the name of the directory into which the chart is expanded
- `--username`

  chart repository username where to locate the requested chart
- `--verify`

  verify the package before using it
- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm registry

Usage:

```
d8 delivery-kit helm registry [options]
```

login to or logout from a registry

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm registry login

Usage:

```
d8 delivery-kit helm registry login [host] [options]
```

login to a registry

Options

- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify registry client using this SSL certificate file
- `--insecure`

  allow connections to TLS registry without certs
- `--key-file`

  identify registry client using this SSL key file
- `--password`, `-p`

  registry password or identity token
- `--password-stdin`

  read password or identity token from stdin
- `--username`, `-u`

  registry username

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH

* interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm registry logout

Usage:

```
d8 delivery-kit helm registry logout [host] [options]
```

logout from a registry

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).

- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm repo

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] [options]
```

add, list, remove, update, and index chart repositories

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

Enable verbose output (default $WERF_LOG_VERBOSE).

- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm repo add

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] add [NAME] [URL] [options]
```

add a chart repository

Options

- `--allow-deprecated-repos`

  by default, this command will not allow adding official repos that have been permanently deleted. This disables that behavior
- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--force-update`

  replace (overwrite) the repo if it already exists
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the repository
- `--key-file`

  identify HTTPS client using this SSL key file
- `--no-update`

  Ignored. Formerly, it would disabled forced updates. It is deprecated by force-update.
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password
- `--password-stdin`

  read chart repository password from stdin
- `--username`

  chart repository username

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)

- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm repo index

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] index [DIR] [options]
```

generate an index file given a directory containing packaged charts

Options

- `--json`

  output in JSON format
- `--merge`

  merge the generated index into the given index
- `--url`

  url of chart repository

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm repo list

Alias: `ls`

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] list [options]
```

list chart repositories

Options

- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm repo remove

Alias: `rm`

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] remove [REPO1 [REPO2 ...]] [options]
```

remove one or more chart repositories

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)

- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm repo update

Alias: `up`

Usage:

```
d8 delivery-kit helm repo add|remove|list|index|update [ARGS] update [REPO1 [REPO2 ...]] [options]
```

update information of available charts locally from chart repositories

Options

- `--fail-on-repo-update-fail`

  update fails if any of the repository updates fail

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm rollback

Usage:

```
d8 delivery-kit helm rollback <RELEASE> [REVISION] [options]
```

roll back a release to a previous revision

Options

- `--cleanup-on-fail`

  allow deletion of new resources created in this rollback when rollback fails
- `--deploy-report-path`

  save deploy report in JSON to the specified path
- `--dry-run`

  simulate a rollback

- `--force`

  force resource update through delete/recreate if needed
- `--history-max`

  limit the maximum number of revisions saved per release. Use 0 for no limit
- `--no-hooks`

  prevent hooks from running during rollback
- `--recreate-pods`

  performs pods restart for the resource if applicable
- `--timeout`

  time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--wait`

  if set, will wait until all Pods, PVCs, Services, and minimum number of Pods of a Deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as --timeout
- `--wait-for-jobs`

  if set and --wait enabled, will wait until all Jobs have been completed before marking the release as successful. It will wait for as long as --timeout

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm search

Usage:

```
d8 delivery-kit helm search [keyword] [options]
```

search for a keyword in charts

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).

- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm search hub

Usage:

```
d8 delivery-kit helm search [keyword] hub [KEYWORD] [options]
```

search for charts in the Artifact Hub or your own hub instance

Options

- `--endpoint`

  Hub instance to query for charts
- `--fail-on-no-result`

  search fails if no results are found
- `--list-repo-url`

  print charts repository URL
- `--max-col-width`

  maximum column width for output table
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm search repo

Usage:

```
d8 delivery-kit helm search [keyword] repo [keyword] [options]
```

search repositories for a keyword in charts

Options

- `--devel`

  use development versions (alpha, beta, and release candidate releases), too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--fail-on-no-result`

  search fails if no results are found
- `--max-col-width`

  maximum column width for output table
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--regexp`, `-r`

  use regular expressions for searching repositories you have added
- `--version`

  search using semantic versioning constraints on repositories you have added
- `--versions`, `-l`

  show the long listing, with each version of each chart on its own line, for repositories you have added

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

Set log color mode.
Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`
  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`
  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`
  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`
  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`
  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`
  namespace scope for this request
- `--quiet`
  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`
  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm secret

Usage:

```
d8 delivery-kit helm secret [options]
```

Work with secrets

Global options

- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`
  No-op
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm secret decrypt

Usage:

```
d8 delivery-kit helm secret decrypt [options]
```

Decrypt data

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.

Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to file instead of stdout
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret encrypt

Usage:

```
d8 delivery-kit helm secret encrypt [options]
```

Encrypt data

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to file instead of stdout
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret file

Usage:

```
d8 delivery-kit helm secret file [options]
```

Work with secret files

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).

- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm secret file decrypt

Usage:

```
d8 delivery-kit helm secret file decrypt [FILE_PATH] [options]
```

Decrypt secret file data

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to file instead of stdout
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret file edit

Usage:

```
d8 delivery-kit helm secret file edit FILE_PATH [options]
```

Edit or create new secret file

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)

- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret file encrypt

Usage:

```
d8 delivery-kit helm secret file encrypt [FILE_PATH] [options]
```

Encrypt file data

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to file instead of stdout
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op

- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret generate-secret-key

Usage:

```
d8 delivery-kit helm secret generate-secret-key [options]
```

Generate hex encryption key

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)

- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret rotate-secret-key

Usage:

```
d8 delivery-kit helm secret rotate-secret-key [EXTRA_SECRET_VALUES_FILE_PATH...] [options]
```

Regenerate secret files with new secret key

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default

$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret values

Usage:

```
d8 delivery-kit helm secret values [options]
```

Work with secret values files

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm secret values decrypt

Usage:

```
d8 delivery-kit helm secret values decrypt [FILE_PATH] [options]
```

Decrypt secret values file data

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to file instead of stdout
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)

- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret values edit

Usage:

```
d8 delivery-kit helm secret values edit FILE_PATH [options]
```

Edit or create new secret values file

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm secret values encrypt

Usage:

```
d8 delivery-kit helm secret values encrypt [FILE_PATH] [options]
```

Encrypt values file data

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)

- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--output-file-path`, `-o`

  Write to file instead of stdout
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

Global options

- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--namespace`, `-n`

  namespace scope for this request
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

d8 delivery-kit helm show

Alias: `inspect`

Usage:

```
d8 delivery-kit helm show [options]
```

show information of a chart

Global options

- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`
  No-op
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`
  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`
  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`
  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`
  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`
  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`
  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`
  namespace scope for this request
- `--quiet`
  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`
  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`
  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm show all

Usage:

```
d8 delivery-kit helm show all [CHART] [options]
```

show all information of the chart

Options

- `--ca-file`
  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`
  identify HTTPS client using this SSL certificate file
- `--devel`
  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--insecure-skip-tls-verify`
  skip tls certificate checks for the chart download
- `--key-file`
  identify HTTPS client using this SSL key file
- `--keyring`
  location of public keys used for verification
- `--pass-credentials`
  pass credentials to all domains
- `--password`
  chart repository password where to locate the requested chart
- `--plain-http`
  use insecure HTTP connections for the chart download
- `--repo`
  chart repository url where to locate the requested chart
- `--username`
  chart repository username where to locate the requested chart
- `--verify`
  verify the package before using it
- `--version`
  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`
  No-op
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`
  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

Disable explanatory output (default $WERF_LOG_QUIET).

- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm show chart

Usage:

```
d8 delivery-kit helm show chart [CHART] [options]
```

show the chart's definition

Options

- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--repo`

  chart repository url where to locate the requested chart
- `--username`

  chart repository username where to locate the requested chart
- `--verify`

  verify the package before using it

- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm show crds

Usage:

```
d8 delivery-kit helm show crds [CHART] [options]
```

show the chart's CRDs

Options

- `--ca-file`
  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`
  identify HTTPS client using this SSL certificate file
- `--devel`
  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--insecure-skip-tls-verify`
  skip tls certificate checks for the chart download
- `--key-file`
  identify HTTPS client using this SSL key file
- `--keyring`
  location of public keys used for verification
- `--pass-credentials`
  pass credentials to all domains
- `--password`
  chart repository password where to locate the requested chart
- `--plain-http`
  use insecure HTTP connections for the chart download
- `--repo`
  chart repository url where to locate the requested chart
- `--username`
  chart repository username where to locate the requested chart
- `--verify`
  verify the package before using it
- `--version`
  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`
  No-op
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`
  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

Disable explanatory output (default $WERF_LOG_QUIET).

- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm show readme

Usage:

```
d8 delivery-kit helm show readme [CHART] [options]
```

show the chart's README

Options

- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--repo`

  chart repository url where to locate the requested chart
- `--username`

  chart repository username where to locate the requested chart
- `--verify`

  verify the package before using it

- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm show values

Usage:

```
d8 delivery-kit helm show values [CHART] [options]
```

show the chart's values

Options

- `--ca-file`
  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`
  identify HTTPS client using this SSL certificate file
- `--devel`
  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--insecure-skip-tls-verify`
  skip tls certificate checks for the chart download
- `--jsonpath`
  supply a JSONPath expression to filter the output
- `--key-file`
  identify HTTPS client using this SSL key file
- `--keyring`
  location of public keys used for verification
- `--pass-credentials`
  pass credentials to all domains
- `--password`
  chart repository password where to locate the requested chart
- `--plain-http`
  use insecure HTTP connections for the chart download
- `--repo`
  chart repository url where to locate the requested chart
- `--username`
  chart repository username where to locate the requested chart
- `--verify`
  verify the package before using it
- `--version`
  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used

Global options

- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`
  No-op
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).

- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.

  Defaults to:

  * $WERF_LOG_TERMINAL_WIDTH

  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm status

Usage:

```
d8 delivery-kit helm status RELEASE_NAME [options]
```

display the status of the named release

Options

- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--revision`

  if set, display the status of the named release with revision
- `--show-desc`

  if set, display the description message of the named release
- `--show-resources`

  if set, display the resources of the named release

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)

- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm template

Usage:

```
d8 delivery-kit helm template [NAME] [CHART] [options]
```

locally render templates

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--api-versions`, `-a`

  Kubernetes api versions used for Capabilities.APIVersions
- `--atomic`

if set, the installation process deletes the installation on failure. The --wait flag will be set automatically if --atomic is used

- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--cleanup-on-fail`

  allow deletion of new resources created in this installation when install fails
- `--create-namespace`

  create the release namespace if not present
- `--dependency-update`

  update dependencies if they are missing before installing the chart
- `--deploy-report-path`

  save deploy report in JSON to the specified path
- `--description`

  add a custom description
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-openapi-validation`

  if set, the installation process will not validate rendered templates against the Kubernetes OpenAPI Schema
- `--dry-run`

  simulate an install. If --dry-run is set with no option being specified or as '--dry-run=client', it will not attempt cluster connections. Setting '--dry-run=server' allows attempting cluster connections.
- `--enable-dns`

  enable DNS lookups when rendering templates
- `--force`

  force resource updates through a replacement strategy
- `--generate-name`, `-g`

  generate the name (and omit the NAME parameter)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--include-crds`

  include CRDs in the templated output
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--is-upgrade`

  set .Release.IsUpgrade instead of .Release.IsInstall
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--kube-version`

  Kubernetes version used for Capabilities.KubeVersion
- `--labels`, `-l`

  Labels that would be added to release metadata. Should be divided by comma.
- `--name-template`

  specify template used to name the release
- `--no-hooks`

  prevent hooks from running during install

- `--output-dir`

  writes the executed templates to files in output-dir instead of stdout
- `--pass-credentials`

  pass credentials to all domains
- `--password`

  chart repository password where to locate the requested chart
- `--plain-http`

  use insecure HTTP connections for the chart download
- `--post-renderer`

  the path to an executable to be used for post rendering. If it exists in $PATH, the binary will be used, otherwise it will try to look for the executable at the given path
- `--post-renderer-args`

  an argument to the post-renderer (can specify multiple)
- `--release-name`

  use release name in the output-dir path.
- `--render-subchart-notes`

  if set, render subchart notes along with the parent
- `--replace`

  re-use the given name, only if that name is a deleted release which remains in the history. This is unsafe in production
- `--repo`

  chart repository url where to locate the requested chart
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  set values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--set-file`

  set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2)
- `--set-json`

  set JSON values on the command line (can specify multiple or separate values with commas: key1=jsonval1,key2=jsonval2)
- `--set-literal`

  set a literal STRING value on the command line
- `--set-string`

  set STRING values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--show-only`, `-s`

  only show manifests rendered from the given templates
- `--skip-crds`

  if set, no CRDs will be installed. By default, CRDs are installed if not already present
- `--skip-tests`

  skip tests from templated output
- `--timeout`

  time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--username`

  chart repository username where to locate the requested chart
- `--validate`

validate your manifests against the Kubernetes cluster you are currently pointing at. This is the same validation performed on an install

- `--values`, `-f`

  specify values in a YAML file or a URL (can specify multiple)
- `--verify`

  verify the package before using it
- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used
- `--wait`

  if set, will wait until all Pods, PVCs, Services, and minimum number of Pods of a Deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as --timeout
- `--wait-for-jobs`

  if set and --wait enabled, will wait until all Jobs have been completed before marking the release as successful. It will wait for as long as --timeout

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm test

Usage:

```
d8 delivery-kit helm test [RELEASE] [options]
```

run tests for a release

Options

- `--filter`

  specify tests by attribute (currently "name") using attribute=value syntax or '!attribute=value' to exclude a test (can specify multiple or separate values with commas: name=test1,name=test2)
- `--logs`

  dump the logs from test pods (this runs after all tests are complete, but before any cleanup)
- `--timeout`

  time to wait for any individual Kubernetes operation (like Jobs for hooks)

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm uninstall

Aliases: `del`, `delete`, `un`

Usage:

```
d8 delivery-kit helm uninstall RELEASE_NAME [...] [options]
```

uninstall a release

Options

- `--cascade`

  Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents. Defaults to background.
- `--description`

  add a custom description
- `--dry-run`

  simulate a uninstall
- `--ignore-not-found`

  Treat "release not found" as a successful uninstall
- `--keep-history`

  remove all associated resources and mark the release as deleted, but retain the release history
- `--no-hooks`

  prevent hooks from running during uninstallation
- `--timeout`

  time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--wait`

  if set, will wait until all the resources are deleted before returning. It will wait for as long as --timeout

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)

- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm upgrade

Usage:

```
d8 delivery-kit helm upgrade [RELEASE] [CHART] [options]
```

upgrade a release

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g.
  $WERF_ADD_ANNOTATION_1=annoName1=annoValue1,
  $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g.
  $WERF_ADD_LABEL_1=labelName1=labelValue1,
  $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--atomic`

if set, upgrade process rolls back changes made in case of failed upgrade. The --wait flag will be set automatically if --atomic is used

- `--ca-file`

  verify certificates of HTTPS-enabled servers using this CA bundle
- `--cert-file`

  identify HTTPS client using this SSL certificate file
- `--cleanup-on-fail`

  allow deletion of new resources created in this upgrade when upgrade fails
- `--create-namespace`

  if --install is set, create the release namespace if not present
- `--dependency-update`

  update dependencies if they are missing before installing the chart
- `--deploy-report-path`

  save deploy report in JSON to the specified path
- `--description`

  add a custom description
- `--devel`

  use development versions, too. Equivalent to version '>0.0.0-0'. If --version is set, this is ignored
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-openapi-validation`

  if set, the upgrade process will not validate rendered templates against the Kubernetes OpenAPI Schema
- `--dry-run`

  simulate an install. If --dry-run is set with no option being specified or as '--dry-run=client', it will not attempt cluster connections. Setting '--dry-run=server' allows attempting cluster connections.
- `--enable-dns`

  enable DNS lookups when rendering templates
- `--force`

  force resource updates through a replacement strategy
- `--history-max`

  limit the maximum number of revisions saved per release. Use 0 for no limit
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-skip-tls-verify`

  skip tls certificate checks for the chart download
- `--install`, `-i`

  if a release by this name doesn't already exist, run an install
- `--key-file`

  identify HTTPS client using this SSL key file
- `--keyring`

  location of public keys used for verification
- `--labels`, `-l`

  Labels that would be added to release metadata. Should be separated by comma. Original release labels will be merged with upgrade labels. You can unset label using null.
- `--no-hooks`

  disable pre/post upgrade hooks
- `--output`, `-o`

  prints the output in the specified format. Allowed values: table, json, yaml
- `--pass-credentials`

  pass credentials to all domains
- `--password`

chart repository password where to locate the requested chart

- `--plain-http`

  use insecure HTTP connections for the chart download
- `--post-renderer`

  the path to an executable to be used for post rendering. If it exists in $PATH, the binary will be used, otherwise it will try to look for the executable at the given path
- `--post-renderer-args`

  an argument to the post-renderer (can specify multiple)
- `--recreate-pods`

  performs pods restart for the resource if applicable
- `--render-subchart-notes`

  if set, render subchart notes along with the parent
- `--repo`

  chart repository url where to locate the requested chart
- `--reset-then-reuse-values`

  when upgrading, reset the values to the ones built into the chart, apply the last release's values and merge in any overrides from the command line via --set and -f. If '--reset-values' or '--reuse-values' is specified, this is ignored
- `--reset-values`

  when upgrading, reset the values to the ones built into the chart
- `--reuse-values`

  when upgrading, reuse the last release's values and merge in any overrides from the command line via --set and -f. If '--reset-values' is specified, this is ignored
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  set values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--set-file`

  set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2)
- `--set-json`

  set JSON values on the command line (can specify multiple or separate values with commas: key1=jsonval1,key2=jsonval2)
- `--set-literal`

  set a literal STRING value on the command line
- `--set-string`

  set STRING values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2)
- `--skip-crds`

  if set, no CRDs will be installed when an upgrade is performed with install flag enabled. By default, CRDs are installed if not already present, when an upgrade is performed with install flag enabled
- `--timeout`

  time to wait for any individual Kubernetes operation (like Jobs for hooks)
- `--username`

  chart repository username where to locate the requested chart
- `--values`, `-f`

  specify values in a YAML file or a URL (can specify multiple)
- `--verify`

verify the package before using it
- `--version`

  specify a version constraint for the chart version to use. This constraint can be a specific tag (e.g. 1.1.1) or it may reference a valid range (e.g. ^2.0.0). If this is not specified, the latest version is used
- `--wait`

  if set, will wait until all Pods, PVCs, Services, and minimum number of Pods of a Deployment, StatefulSet, or ReplicaSet are in a ready state before marking the release as successful. It will wait for as long as --timeout
- `--wait-for-jobs`

  if set and --wait enabled, will wait until all Jobs have been completed before marking the release as successful. It will wait for as long as --timeout

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm verify

Usage:

```
d8 delivery-kit helm verify PATH [options]
```

verify that a chart at the given path has been signed and is valid

Options

- `--keyring`

  keyring containing public keys

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds

- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit helm version

Usage:

```
d8 delivery-kit helm version [options]
```

print the client version information

Options

- `--client`, `-c`

  display client version information
- `--short`

  print the version number
- `--template`

  template for version string format

Global options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--hooks-status-progress-period`

  No-op
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).

- `--namespace`, `-n`

  namespace scope for this request
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to $WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit host

Usage:

```
d8 delivery-kit host [options]
```

Work with werf cache and data of all projects on the host machine

d8 delivery-kit host cleanup

Usage:

```
d8 delivery-kit host cleanup [options]
```

Cleanup old unused werf cache and data of all projects on host machine.

Options

- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)

- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--force`

  Force deletion of images which are being used by some containers (default $WERF_FORCE)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).

- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit host purge

Usage:

```
d8 delivery-kit host purge [options]
```

Purge werf images, cache and other data for all projects on host machine

Options

- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--force`

  First remove containers that use werf docker images which are going to be deleted
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit includes

Usage:

```
d8 delivery-kit includes [options]
```

Work with werf includes

d8 delivery-kit includes get-file

Usage:

```
d8 delivery-kit includes get-file [FILE_NAME] [options]
```

Display file content that will be used by werf according to the includes overlay rules.

Options

- `--allow-includes-update`
  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`
  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-templates-dir`
  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--dev`
  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`
  Use specified environment (default $WERF_ENV)
- `--follow`
  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`
  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`
  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`
  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`
  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`
  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`
  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

Disable explanatory output (default $WERF_LOG_QUIET).

- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit includes ls-files

Usage:

```
d8 delivery-kit includes ls-files [GLOB...] [options]
```

List files in the project according to the includes overlay rules.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).

Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)

- `--dir`

  Use specified project directory where project's werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--filter`

  Filter by source, e.g. --filter=source=local,remoteRepo (default $WERF_INCLUDES_LIST_FILTER or all sources).
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout's file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--ssh-key`

Use only specific ssh key(s).
Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}

- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit includes update

Usage:

```
d8 delivery-kit includes update [options]
```

Create or update includes lock file (default: werf-includes.lock).

Options

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--log-color-mode`

  Set log color mode.

Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
Default $WERF_LOG_COLOR_MODE or auto mode.

- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit kube-run

Usage:

```
d8 delivery-kit kube-run [options] [IMAGE_NAME] [-- COMMAND ARG...]
```

Run container for project image in Kubernetes

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).

Format: labelName=labelValue.
Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--auto-pull-secret`

  Automatically create docker config secret in the namespace and plug it via pod's imagePullSecrets for private registry access (default $WERF_AUTO_PULL_SECRET or true if not specified)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--copy-from`

  Copy file/dir from container to local machine after user command execution. Example: "/from/file:to". Can be specified multiple times. Can also be defined with "$WERF_COPY_FROM_*", e.g. "WERF_COPY_FROM_1=from:to".
- `--copy-to`

  Copy file/dir from local machine to container before user command execution. Example: "from:/to/file". Can be specified multiple times. Can also be defined with "$WERF_COPY_TO_*", e.g. "WERF_COPY_TO_1=from:to".
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--extra-options`

  Pass extra options to "kubectl run" command, which will create a Pod (default $WERF_EXTRA_OPTIONS)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree

- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--interactive`, `-i`

  Enable interactive mode (default $WERF_INTERACTIVE or false if not specified)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--overrides`

  Inline JSON to override/extend any fields in created Pod, e.g. to add imagePullSecrets field (default $WERF_OVERRIDES). %pod_name%, %container_name%, and %container_image% will be replaced with the names of the created pod, container, and container image, respectively.
- `--platform`

Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)

- `--pod`

  Set created pod name (default $WERF_POD or autogenerated if not specified)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--rm`

  Remove pod and other created resources after command completion (default $WERF_RM or true if not specified)
- `--rm-with-namespace`

  Remove also a namespace after command completion (default $WERF_RM_WITH_NAMESPACE or false if not specified)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)

- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default
  $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa,
  $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--tty`, `-t`

  Allocate a TTY (default $WERF_TTY or false if not specified)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)

d8 delivery-kit lint

Usage:

```
d8 delivery-kit lint [IMAGE_NAME...] [options]
```

Lint Helm chart

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g.
  $WERF_ADD_ANNOTATION_1=annoName1=annoValue1,
  $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).

Format: labelName=labelValue.
Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--delete-propagation`

  Set default delete propagation strategy (default $WERF_DELETE_PROPAGATION or Foreground).
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default

$WERF_DISABLE_DEFAULT_VALUES or false)

- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo and to pull base images
- `--env`

  Use specified environment (default $WERF_ENV)
- `--extra-apiversions`

  Extra Kubernetes API versions passed to $.Capabilities.APIVersions. Can be also set with $WERF_EXTRA_APIVERSIONS_* environment variables, values can be comma-separated
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default

$WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)

- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--kube-version`

  Set specific Capabilities.KubeVersion (default $WERF_KUBE_VERSION)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

Enable verbose output (default $WERF_LOG_VERBOSE).

- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or $WERF_RELEASE)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)

- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--runtime-annotations`

  Add annotations which will not trigger resource updates to all resources (default $WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`

  Add labels which will not trigger resource updates to all resources (default $WERF_RUNTIME_LABELS)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path

and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)

- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--use-custom-tag`

Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED). The alias may contain the following shortcuts:
- %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
- %image_content_based_tag% to use a content-based tag.
For cleaning custom tags and associated content-based tag are treated as one.
Also, can be defined with $WERF_USE_CUSTOM_TAG (e.g. $WERF_USE_CUSTOM_TAG="%image%-tag")

- `--validate`

  Validate your manifests against the Kubernetes cluster you are currently pointing at (default $WERF_VALIDATE)
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit managed-images

Usage:

```
d8 delivery-kit managed-images [options]
```

Work with managed images which will be preserved during cleanup procedure

d8 delivery-kit managed-images add

Usage:

```
d8 delivery-kit managed-images add [options]
```

Add image record to the list of managed images which will be preserved during cleanup procedure.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in

working directory)
- `--container-registry-mirror`
  (Buildah-only) Use specified mirrors for docker.io
- `--debug`
  Enable debug (default $WERF_DEBUG).
- `--debug-templates`
  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`
  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`
  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`
  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`
  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`
  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and write images to the specified repo
- `--env`
  Use specified environment (default $WERF_ENV)
- `--final-repo`
  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`
  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`
  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`
  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`
  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`
  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`
  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`
  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`
  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`
  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`
  Custom path to the giterminism configuration file relative to working directory (default

$WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit managed-images ls

Usage:

```
d8 delivery-kit managed-images ls [options]
```

List managed images which will be preserved during cleanup procedure.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before

making requests to the primary repo.
Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)

- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read images from the specified repo
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)

- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).

- `--repo`
  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`
  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`
  Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-registry`
  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`
  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`
  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`
  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`
  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit managed-images rm

Usage:

```
d8 delivery-kit managed-images rm [options]
```

Remove image record from the list of managed images which will be preserved during cleanup procedure.

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and write images to the specified repo
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)

- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).

- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local

address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit plan

Usage:

```
d8 delivery-kit plan [options]
```

Prepare deploy plan and show how resources in a Kubernetes cluster would change on next deploy

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g.
  $WERF_ADD_ANNOTATION_1=annoName1=annoValue1,
  $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-custom-tag`

  Set tag alias for the content-based tag.
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also can be defined with $WERF_ADD_CUSTOM_TAG_* (e.g.
  $WERF_ADD_CUSTOM_TAG_1="%image%-tag1", $WERF_ADD_CUSTOM_TAG_2="%image%-tag2")
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g.
  $WERF_ADD_LABEL_1=labelName1=labelValue1,
  $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or

$WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)

- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--delete-propagation`

  Set default delete propagation strategy (default $WERF_DELETE_PROPAGATION or Foreground).
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--deploy-graph-path`

  Save deploy graph path to the specified file (by default $WERF_DEPLOY_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--dev`

Enable development mode (default $WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--diff-context-lines`

  Show N lines of context around diffs ($WERF_DIFF_CONTEXT_LINES by default)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--exit-code`

  Return exit code 0 if no changes, 1 if error, 2 if any changes planned (default $WERF_EXIT_CODE or false)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)

- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.
  In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--kube-api-server`
  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`
  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`
  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`
  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`
  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`
  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`
  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`
  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`
  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`
  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`
  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`
  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`
  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`
  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`
  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`
  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`
  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default

$WERF_KUBE_KEY_DATA)

- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`

  By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)
- `--no-install-crds`

  Do not install CRDs from "crds/" directories of installed charts (default $WERF_NO_INSTALL_CRDS)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the

manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)

- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or $WERF_RELEASE)
- `--release-info-annotations`

  Add annotations to release metadata (default $WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`

  Add Helm release labels (can specify multiple). Kind of labels depends or release storage driver. Format: labelName=labelValue.
  Also, can be specified with $WERF_RELEASE_LABEL_* (e.g. $WERF_RELEASE_LABEL_1=labelName1=labelValue1, $WERF_RELEASE_LABEL_2=labelName2=labelValue2)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)

- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--runtime-annotations`

  Add annotations which will not trigger resource updates to all resources (default $WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`

  Add labels which will not trigger resource updates to all resources (default $WERF_RUNTIME_LABELS)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--save-plan`

  Save the gzip-compressed JSON install plan to the specified file
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1, $WERF_SET_FILE_2=key2=val2)
- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with

commas: key1=val1,key2=val2.
Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)

- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing.
  Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--show-insignificant-diffs`

  Show insignificant diff lines ($WERF_SHOW_INSIGNIFICANT_DIFFS by default)
- `--show-plan`

  Show plan artifact planned changes
- `--show-sensitive-diffs`

  Show sensitive diff lines ($WERF_SHOW_SENSITIVE_DIFFS by default)
- `--show-verbose-crd-diffs`

  Show verbose CRD diff lines ($WERF_SHOW_VERBOSE_CRD_DIFFS by default)
- `--show-verbose-diffs`

  Show verbose diff lines ($WERF_SHOW_VERBOSE_DIFFS by default)
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-helm-dependencies`

  Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default

$WERF_SKIP_TLS_VERIFY_REGISTRY)

- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--status-progress-period`

  No-op
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--timeout`, `-t`

  No-op
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--use-custom-tag`

  Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also, can be defined with $WERF_USE_CUSTOM_TAG (e.g. $WERF_USE_CUSTOM_TAG="%image%-tag")
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit purge

Usage:

```
d8 delivery-kit purge [options]
```

Purge all project images in the container registry

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to delete images from the specified repo
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)

- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)

- `--quiet`
  Disable explanatory output (default $WERF_QUIET).
- `--repo`
  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`
  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`
  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`
  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`
  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`
  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`
  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`
  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`
  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--secondary-repo`
  Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-tls-verify-registry`
  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--synchronization`, `-S`
  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`
  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`
  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit render

Usage:

```
d8 delivery-kit render [IMAGE_NAME...] [options]
```

Render Kubernetes templates

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--deno-binary-path`

  Path to the Deno binary to use instead of auto-downloading (default $WERF_DENO_BINARY_PATH)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")

- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-default-secret-values`

  Do not use secret values from the default .helm/secret-values.yaml file (default $WERF_DISABLE_DEFAULT_SECRET_VALUES or false)
- `--disable-default-values`

  Do not use values from the default .helm/values.yaml file (default $WERF_DISABLE_DEFAULT_VALUES or false)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo and to pull base images
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--extra-apiversions`

  Extra Kubernetes API versions passed to $.Capabilities.APIVersions. Can be also set with $WERF_EXTRA_APIVERSIONS_* environment variables, values can be comma-separated
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or true by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)

- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--ignore-bundle-js`

  Do not use the existing bundle.js file. Requires TypeScript source files and Deno to rebuild (default $WERF_IGNORE_BUNDLE_JS)
- `--ignore-secret-key`

  Disable secrets decryption (default $WERF_IGNORE_SECRET_KEY)
- `--include-crds`

  Include CRDs in the templated output (default $WERF_INCLUDE_CRDS)
- `--insecure-helm-dependencies`

  Allow insecure oci registries to be used in the Chart.yaml dependencies configuration (default $WERF_INSECURE_HELM_DEPENDENCIES)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)

- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--kube-version`

Set specific Capabilities.KubeVersion (default $WERF_KUBE_VERSION)

- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--output`

  Write render output to the specified file instead of stdout ($WERF_RENDER_OUTPUT by default)
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--provenance-keyring`

  Path to keyring containing public keys to verify chart provenance (default $WERF_PROVENANCE_KEYRING)
- `--provenance-strategy`

  Strategy for provenance verifying (default $WERF_PROVENANCE_STRATEGY).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or $WERF_RELEASE)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)

- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--secret-key`

  Secret key (default $WERF_SECRET_KEY)
- `--secret-values`

  Specify helm secret values in a YAML file (can specify multiple). Also, can be defined with $WERF_SECRET_VALUES_* (e.g. $WERF_SECRET_VALUES_ENV=.helm/secret_values_test.yaml, $WERF_SECRET_VALUES_DB=.helm/secret_values_db.yaml)
- `--set`

  Set helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_* (e.g. $WERF_SET_1=key1=val1, $WERF_SET_2=key2=val2)
- `--set-docker-config-json-value`

  Shortcut to set current docker config into the .Values.dockerconfigjson
- `--set-file`

  Set values from respective files specified via the command line (can specify multiple or separate values with commas: key1=path1,key2=path2).
  Also, can be defined with $WERF_SET_FILE_* (e.g. $WERF_SET_FILE_1=key1=path1,

$WERF_SET_FILE_2=key2=val2)

- `--set-json`

  Set new values, where the key is the value path and the value is JSON (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_JSON_* (e.g. $WERF_SET_JSON_1=key1=val1, $WERF_SET_JSON_2=key2=val2)
- `--set-literal`

  Set new values, where the key is the value path and the value is the value. The value will always become a literal string (can specify multiple or separate values with commas: key1=val1,key2=val2).)
  Also, can be defined with $WERF_SET_LITERAL_* (e.g. $WERF_SET_LITERAL_1=key1=val1, $WERF_SET_LITERAL_2=key2=val2)
- `--set-root-json`

  Set new keys in arbitrary things in the global root context ("$"), where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you REALLY know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_ROOT_JSON_* (e.g. $WERF_SET_ROOT_JSON_1=key1=val1, $WERF_SET_ROOT_JSON_2=key2=val2)
- `--set-runtime-json`

  Set new keys in $.Runtime, where the key is the value path and the value is JSON. This is meant to be generated inside the program, so use --set-json instead, unless you know what you are doing. Can specify multiple or separate values with commas: key1=val1,key2=val2.
  Also, can be defined with $WERF_SET_RUNTIME_JSON_* (e.g. $WERF_SET_RUNTIME_JSON_1=key1=val1, $WERF_SET_RUNTIME_JSON_2=key2=val2)
- `--set-string`

  Set STRING helm values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
  Also, can be defined with $WERF_SET_STRING_* (e.g. $WERF_SET_STRING_1=key1=val1, $WERF_SET_STRING_2=key2=val2)
- `--show-only`, `-s`

  only show manifests rendered from the given templates
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-dependencies-repo-refresh`, `-L`

  Do not refresh helm chart repositories locally cached index
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-helm-dependencies`

Skip TLS certificate validation when accessing a Helm charts repository (default $WERF_SKIP_TLS_VERIFY_HELM_DEPENDENCIES)

- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--status-progress-period`

  No-op
- `--stub-tags`

  Use stubs instead of real tags (default $WERF_STUB_TAGS)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--templates-allow-dns`

  Allow performing DNS requests in templating (default $WERF_TEMPLATES_ALLOW_DNS)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--use-custom-tag`

  Use a tag alias in helm templates instead of an image content-based tag (NOT RECOMMENDED).
  The alias may contain the following shortcuts:
  - %image%, %image_slug% or %image_safe_slug% to use the image name (necessary if there is more than one image in the werf config);
  - %image_content_based_tag% to use a content-based tag.
  For cleaning custom tags and associated content-based tag are treated as one.
  Also, can be defined with $WERF_USE_CUSTOM_TAG (e.g. $WERF_USE_CUSTOM_TAG="%image%-tag")
- `--validate`

  Validate your manifests against the Kubernetes cluster you are currently pointing at (default $WERF_VALIDATE)
- `--values`

  Specify helm values in a YAML file or a URL (can specify multiple). Also, can be defined with $WERF_VALUES_* (e.g. $WERF_VALUES_1=.helm/values_1.yaml, $WERF_VALUES_2=.helm/values_2.yaml)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state

($WERF_VIRTUAL_MERGE by default)
- `--without-images`

  Disable building of images defined in the werf.yaml (if any) and usage of such images in the .helm/templates ($WERF_WITHOUT_IMAGES or false by default — e.g. enable all images defined in the werf.yaml by default)

d8 delivery-kit rollback

Usage:

```
d8 delivery-kit rollback [options]
```

Rollback the Helm release

Options

- `--add-annotation`

  Add annotation to deploying resources (can specify multiple).
  Format: annoName=annoValue.
  Also, can be specified with $WERF_ADD_ANNOTATION_* (e.g. $WERF_ADD_ANNOTATION_1=annoName1=annoValue1, $WERF_ADD_ANNOTATION_2=annoName2=annoValue2)
- `--add-label`

  Add label to deploying resources (can specify multiple).
  Format: labelName=labelValue.
  Also, can be specified with $WERF_ADD_LABEL_* (e.g. $WERF_ADD_LABEL_1=labelName1=labelValue1, $WERF_ADD_LABEL_2=labelName2=labelValue2)
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--delete-propagation`

  Set default delete propagation strategy (default $WERF_DELETE_PROPAGATION or Foreground).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--env`

  Use specified environment (default $WERF_ENV)

- `--force-adoption`

  Always adopt resources, even if they belong to a different Helm release (default $WERF_FORCE_ADOPTION or false)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--hooks-status-progress-period`

  No-op
- `--kube-api-server`

  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`

  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`

  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`

  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`

  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`

  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`

  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`

  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`

  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`

  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`

  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`

  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`

  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`

  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)

- `--kube-impersonate-user`

  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`

  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`

  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

  Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)
- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--namespace`

  Use specified Kubernetes namespace (default [[project]]-[[env]] template or deploy.namespace custom template from werf.yaml or $WERF_NAMESPACE)
- `--network-parallelism`

  Parallelize some network operations (default $WERF_NETWORK_PARALLELISM or 30)
- `--no-final-tracking`

By default disable tracking operations that have no create/update/delete resource operations after them, which are most tracking operations, to speed up the release (default $WERF_NO_FINAL_TRACKING)

- `--no-notes`

  Don't show release notes at the end of the release (default $WERF_NO_NOTES)
- `--no-pod-logs`

  Disable Pod logs collection and printing (default $WERF_NO_POD_LOGS or false)
- `--no-remove-manual-changes`

  Don't remove fields added manually to the resource in the cluster if fields aren't present in the manifest (default $WERF_NO_REMOVE_MANUAL_CHANGES)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--release`

  Use specified Helm release name (default [[project]]-[[env]] template or deploy.helmRelease custom template from werf.yaml or $WERF_RELEASE)
- `--release-info-annotations`

  Add annotations to release metadata (default $WERF_RELEASE_INFO_ANNOTATIONS)
- `--release-label`

  Add Helm release labels (can specify multiple). Kind of labels depends or release storage driver. Format: labelName=labelValue.
  Also, can be specified with $WERF_RELEASE_LABEL_* (e.g. $WERF_RELEASE_LABEL_1=labelName1=labelValue1, $WERF_RELEASE_LABEL_2=labelName2=labelValue2)
- `--release-storage`

  How releases should be stored (default $WERF_RELEASE_STORAGE)
- `--release-storage-sql-connection`

  SQL Connection String for Helm SQL Storage (default $WERF_RELEASE_STORAGE_SQL_CONNECTION)
- `--releases-history-max`

  Max releases to keep in release storage ($WERF_RELEASES_HISTORY_MAX or 5 by default)
- `--revision`

  Revision number to rollback to (if not specified, rolls back to previous revision)
- `--rollback-graph-path`

  Save rollback graph path to the specified file (by default $WERF_ROLLBACK_GRAPH_PATH). Extension must be .dot or not specified. If extension not specified, then .dot is used
- `--rollback-report-path`

  Change rollback report path and format (by default $WERF_ROLLBACK_REPORT_PATH or ".werf-rollback-report.json" if not set). Extension must be .json for JSON format. If extension not specified, then .json is used
- `--runtime-annotations`

  Add annotations which will not trigger resource updates to all resources (default $WERF_RUNTIME_ANNOTATIONS)
- `--runtime-labels`

  Add labels which will not trigger resource updates to all resources (default $WERF_RUNTIME_LABELS)
- `--save-rollback-report`

  Save rollback report (by default $WERF_SAVE_ROLLBACK_REPORT or false). Its path and format configured with --rollback-report-path
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--status-progress-period`

  Status progress period in seconds. Set -1 to stop showing status progress. Defaults to

$WERF_STATUS_PROGRESS_PERIOD_SECONDS or 5 seconds
- `--timeout`, `-t`

  Resources tracking timeout in seconds ($WERF_TIMEOUT by default)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit run

Usage:

```
d8 delivery-kit run [options] [IMAGE_NAME] [-- COMMAND ARG...]
```

Run container for project image

Options

- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--annotate-layers-with-dm-verity-root-hash`

  Enable annotation of image layers with dm-verity root hash (default $WERF_ANNOTATE_LAYERS_WITH_DM_VERITY_ROOT_HASH)
- `--backend-network`

  Network mode for the build containers ($WERF_BACKEND_NETWORK or nothing by default)
- `--bash`

  Use predefined docker options and command for debug
- `--bsign-elf-files`

  Enable ELF files signing with bsign (default $WERF_BSIGN_ELF_FILES).
  When enabled, the private elf key must be specified with --elf-pgp-private-key-base64 or --elf-pgp-private-key-fingerprint option
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

Enable development mode (default $WERF_DEV).
The mode allows working with project files without doing redundant commits during debugging and development

- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read and pull images from the specified repo
- `--docker-options`

  Define docker run options (default $WERF_DOCKER_OPTIONS)
- `--dry-run`

  Indicate what the command would do without actually doing that (default $WERF_DRY_RUN)
- `--elf-pgp-private-key-base64`

  Base64-encoded PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_BASE64)
- `--elf-pgp-private-key-fingerprint`

  PGP private key fingerprint (default $WERF_ELF_PGP_PRIVATE_KEY_FINGERPRINT)
- `--elf-pgp-private-key-passphrase`

  Passphrase for the PGP private key (default $WERF_ELF_PGP_PRIVATE_KEY_PASSPHRASE)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--follow`

  Enable follow mode (default $WERF_FOLLOW).
  The mode allows restarting the command on a new commit.

In development mode (--dev), werf restarts the command on any changes (including untracked files) in the git repository worktree
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-helm-dependencies`

  No-op
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

Container registry storage address (default $WERF_REPO)

- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--require-built-images`, `-Z`

  Requires all used images to be previously built and exist in repo. Exits with error if needed images are not cached and so require to run build instructions (default $WERF_REQUIRE_BUILT_IMAGES)
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--shell`

  Use predefined docker options and command for debug
- `--sign-cert`

  The leaf certificate as path to PEM file or base64-encoded PEM (default $WERF_SIGN_CERT)
- `--sign-elf-files`

  Enable ELF files signing (default $WERF_SIGN_ELF_FILES).
  When enabled, the private signing key must be specified with --sign-key option and the certificate must be specified with --sign-cert option
- `--sign-intermediates`

  The intermediate certificates as path to PEM file or base64-encoded PEM (default $WERF_SIGN_INTERMEDIATES)
- `--sign-key`

  The private signing key as path to PEM file, base64-encoded PEM or hashivault://[KEY] (default $WERF_SIGN_KEY)
- `--sign-manifest`

  Enable image manifest signing (default $WERF_SIGN_MANIFEST).
  When enabled,
  the private signing key must be specified with --sign-key option and
  the certificate must be specified with --sign-cert option
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

Use only specific ssh key(s).
Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}

- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)

d8 delivery-kit sbom

Usage:

```
d8 delivery-kit sbom [options]
```

Work with werf SBOM images

d8 delivery-kit sbom get

Usage:

```
d8 delivery-kit sbom get [IMAGE_NAME] [options]
```

Get SBOM of an image

Options

- `--allowed-backend-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE)
- `--allowed-backend-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_BACKEND_STORAGE_VOLUME_USAGE_MARGIN)
- `--allowed-docker-storage-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of backend (Docker or Buildah) storage volume usage which will cause cleanup of least recently used local backend images (default 70% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE)
- `--allowed-docker-storage-volume-usage-margin`

  During cleanup of least recently used local backend (Docker or Buildah) images werf would delete

images until volume usage becomes below "allowed-backend-storage-volume-usage - allowed-backend-storage-volume-usage-margin" level (default 5% or $WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN)

- `--allowed-local-cache-volume-usage`

  Set allowed percentage or absolute value (e.g. 10GB) of local cache (~/.werf/local_cache by default) volume usage which will cause cleanup of least recently used data from the local cache (default 70% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE)
- `--allowed-local-cache-volume-usage-margin`

  During cleanup of local cache werf would delete local cache data until volume usage becomes below "allowed-local-cache-volume-usage - allowed-local-cache-volume-usage-margin" level (default 5% or $WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If extension not specified, then .json is used
- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple). Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--digest`

  Digest of the image to get SBOM for (mutually exclusive with --tag)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--disable-auto-host-cleanup`

  Disable auto host cleanup procedure in main werf commands like werf-build, werf-converge and other (default disabled or WERF_DISABLE_AUTO_HOST_CLEANUP)
- `--docker-config`

Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
Command needs granted permissions to read, pull and push images into the specified repo and to pull base images

- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image

specified with ~.
STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec

- `--kube-api-server`
  Kubernetes API server address (default $WERF_KUBE_API_SERVER)
- `--kube-auth-password`
  Basic auth password for Kubernetes API (default $WERF_KUBE_AUTH_PASSWORD)
- `--kube-auth-provider`
  Auth provider name for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER)
- `--kube-auth-provider-config`
  Auth provider config for authentication in Kubernetes API (default $WERF_KUBE_AUTH_PROVIDER_CONFIG)
- `--kube-auth-username`
  Basic auth username for Kubernetes API (default $WERF_KUBE_AUTH_USERNAME)
- `--kube-burst-limit`
  Kubernetes client burst limit (default $WERF_KUBE_BURST_LIMIT or 100)
- `--kube-ca-data`
  Pass Kubernetes API server TLS CA data (default $WERF_KUBE_CA_DATA)
- `--kube-ca-path`
  Kubernetes API server CA path (default $WERF_KUBE_CA_PATH)
- `--kube-cert`
  Path to PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT
- `--kube-cert-data`
  Pass PEM-encoded TLS client cert for connecting to Kubernetes API (default $WERF_KUBE_CERT_DATA)
- `--kube-config`
  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`
  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`
  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kube-context-cluster`
  Use cluster from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_CLUSTER)
- `--kube-context-user`
  Use user from Kubeconfig for current context (default $WERF_KUBE_CONTEXT_USER)
- `--kube-impersonate-group`
  Sets Impersonate-Group headers when authenticating in Kubernetes. Can be also set with $WERF_KUBE_IMPERSONATE_GROUP_* environment variables
- `--kube-impersonate-uid`
  Sets Impersonate-Uid header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_UID)
- `--kube-impersonate-user`
  Sets Impersonate-User header when authenticating in Kubernetes (default $WERF_KUBE_IMPERSONATE_USER)
- `--kube-key`
  Path to PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY)
- `--kube-key-data`
  Pass PEM-encoded TLS client key for connecting to Kubernetes API (default $WERF_KUBE_KEY_DATA)
- `--kube-proxy-url`

Proxy URL to use for proxying all requests to Kubernetes API (default $WERF_KUBE_PROXY_URL)

- `--kube-qps-limit`

  Kubernetes client QPS limit (default $WERF_KUBE_QPS_LIMIT or 30)
- `--kube-request-timeout`

  Timeout for all requests to Kubernetes API (default $WERF_KUBE_REQUEST_TIMEOUT)
- `--kube-tls-server`

  Server name to use for Kubernetes API server certificate validation. If it is not provided, the hostname used to contact the server is used (default $WERF_KUBE_TLS_SERVER)
- `--kube-token`

  Kubernetes bearer token used for authentication (default $WERF_KUBE_TOKEN)
- `--kube-token-path`

  Path to file with bearer token for authentication in Kubernetes (default $WERF_KUBE_TOKEN_PATH)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.

The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache. Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-kube`

  Skip TLS certificate validation when accessing a Kubernetes cluster (default $WERF_SKIP_TLS_VERIFY_KUBE)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--ssh-key`

  Use only specific ssh key(s).
  Can be specified with $WERF_SSH_KEY_* (e.g. $WERF_SSH_KEY_REPO=~/.ssh/repo_rsa, $WERF_SSH_KEY_NODEJS=~/.ssh/nodejs_rsa).
  Defaults to $WERF_SSH_KEY_*, system ssh-agent or ~/.ssh/{id_rsa|id_dsa}
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tag`

  Content-based tag of the image to get SBOM for (mutually exclusive with --digest)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

- `--virtual-merge`

  Enable virtual/ephemeral merge commit mode when building current application state ($WERF_VIRTUAL_MERGE by default)

d8 delivery-kit sbom merge

Usage:

```
d8 delivery-kit sbom merge [options]
```

Merge per-image SBOMs into a product-level SBOM

Options

- `--app-name`

  Application/product name for the merged SBOM metadata
- `--app-version`

  Application/product version for the merged SBOM metadata
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to pull SBOM images from the specified repo
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--input`

  Path to JSON mapping file (image name -> sha256:digest)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--ispras-format`

  ISPRAS SBOM format: "oss" or "container"
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--manufacturer`

Manufacturer name for the merged SBOM metadata
- `--output`, `-o`

  Output file path (defaults to stdout)
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit sbom validate

Usage:

```
d8 delivery-kit sbom validate [options]
```

Validate SBOM file against ISPRAS schemas

Options

- `--check-vcs`

  Enable VCS URL validation
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--docker-config`

Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
Command needs granted permissions to pull the ISPRAS SBOM checker image

- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--ispras-format`

  ISPRAS SBOM format: oss or container
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--path`

  Path to CycloneDX JSON SBOM file (repeatable)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit slugify

Usage:

```
d8 delivery-kit slugify STRING [options]
```

Print slugged string by specified format.

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--format`, `-f`

  r|helm-release: suitable for Helm Release

ns|kubernetes-namespace: suitable for Kubernetes Namespace
tag|docker-tag: suitable for Docker Tag

- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit stages

Usage:

```
d8 delivery-kit stages [options]
```

Work with werf stages

d8 delivery-kit stages copy

Usage:

```
d8 delivery-kit stages copy [options]
```

Copy stages between container registry and archive

Options

- `--all`

  Copy all project stages (default: true). Use --all=false to copy stages for current build only. Note: this flag is ignored when copying from archive to container registry.
- `--allow-includes-update`

  Allow use includes latest versions (default $WERF_ALLOW_INCLUDES_UPDATE or false)
- `--backend-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_BACKEND_STORAGE_PATH)
- `--build-report-path`

  Change build report path and format (by default $WERF_BUILD_REPORT_PATH or ".werf-build-report.json" if not set). Extension must be either .json for JSON format or .env for env-file format. If

extension not specified, then .json is used

- `--cache-repo`

  Specify one or multiple cache repos with images that will be used as a cache. Cache will be populated when pushing newly built images into the primary repo and when pulling existing images from the primary repo. Cache repo will be used to pull images and to get manifests before making requests to the primary repo.
  Also, can be specified with $WERF_CACHE_REPO_* (e.g. $WERF_CACHE_REPO_1=..., $WERF_CACHE_REPO_2=...)
- `--config`

  Use custom configuration file (default $WERF_CONFIG or werf.yaml in the project directory)
- `--config-render-path`

  Custom path for storing rendered configuration file
- `--config-templates-dir`

  Custom configuration templates directory (default $WERF_CONFIG_TEMPLATES_DIR or .werf in working directory)
- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--debug-templates`

  Enable debug mode for Go templates (default $WERF_DEBUG_TEMPLATES or false)
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--dir`

  Use specified project directory where project’s werf.yaml and other configuration files should reside (default $WERF_DIR or current working directory)
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repos
- `--docker-server-storage-path`

  Use specified path to the local backend (Docker or Buildah) storage to check backend storage volume usage while performing garbage collection of local backend images (detect local backend storage path by default or use $WERF_DOCKER_SERVER_STORAGE_PATH)
- `--env`

  Use specified environment (default $WERF_ENV)
- `--final-images-only`

  Process final images only ($WERF_FINAL_IMAGES_ONLY or false by default)
- `--final-repo`

  Container registry storage address (default $WERF_FINAL_REPO)
- `--final-repo-container-registry`

  Choose final-repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_FINAL_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).

- `--final-repo-docker-hub-password`

  final-repo Docker Hub password (default $WERF_FINAL_REPO_DOCKER_HUB_PASSWORD)
- `--final-repo-docker-hub-token`

  final-repo Docker Hub token (default $WERF_FINAL_REPO_DOCKER_HUB_TOKEN)
- `--final-repo-docker-hub-username`

  final-repo Docker Hub username (default $WERF_FINAL_REPO_DOCKER_HUB_USERNAME)
- `--final-repo-github-token`

  final-repo GitHub token (default $WERF_FINAL_REPO_GITHUB_TOKEN)
- `--final-repo-harbor-password`

  final-repo Harbor password (default $WERF_FINAL_REPO_HARBOR_PASSWORD)
- `--final-repo-harbor-username`

  final-repo Harbor username (default $WERF_FINAL_REPO_HARBOR_USERNAME)
- `--final-repo-quay-token`

  final-repo quay.io token (default $WERF_FINAL_REPO_QUAY_TOKEN)
- `--from`

  Source address to copy stages from. Use archive:PATH for stage archive or [docker://]REPO for container registry.
- `--git-work-tree`

  Use specified git work tree dir (default $WERF_WORK_TREE or lookup for directory that contains .git in the current or parent directories)
- `--giterminism-config`

  Custom path to the giterminism configuration file relative to working directory (default $WERF_GITERMINISM_CONFIG or werf-giterminism.yaml in working directory)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--introspect-before-error`

  Introspect failed stage in the clean state, before running all assembly instructions of the stage
- `--introspect-error`

  Introspect failed stage in the state, right after running failed assembly instruction
- `--introspect-stage`

  Introspect a specific stage. The option can be used multiple times to introspect several stages.

  There are the following formats to use:
  * specify IMAGE_NAME/STAGE_NAME to introspect stage STAGE_NAME of either image or artifact IMAGE_NAME
  * specify STAGE_NAME or */STAGE_NAME for the introspection of all existing stages with name STAGE_NAME

  IMAGE_NAME is the name of an image or artifact described in werf.yaml, the nameless image specified with ~.
  STAGE_NAME should be one of the following: from, beforeInstall, dependenciesBeforeInstall, gitArchive, install, dependenciesAfterInstall, beforeSetup, dependenciesBeforeSetup, setup, dependenciesAfterSetup, gitCache, gitLatestPatch, dockerInstructions, dockerfile, imageSpec
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).

- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)
- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--platform`

  Enable platform emulation when building images with werf, format: OS/ARCH[/VARIANT] ($WERF_PLATFORM or $DOCKER_DEFAULT_PLATFORM by default)
- `--project-name`, `-N`

  Set a specific project name (default $WERF_PROJECT_NAME)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--repo`

  Container registry storage address (default $WERF_REPO)
- `--repo-container-registry`

  Choose repo container registry implementation.
  The following container registries are supported: ecr, acr, default, dockerhub, gcr, github, gitlab, harbor, quay.
  Default $WERF_REPO_CONTAINER_REGISTRY or auto mode (detect container registry by repo address).
- `--repo-docker-hub-password`

  repo Docker Hub password (default $WERF_REPO_DOCKER_HUB_PASSWORD)
- `--repo-docker-hub-token`

  repo Docker Hub token (default $WERF_REPO_DOCKER_HUB_TOKEN)
- `--repo-docker-hub-username`

  repo Docker Hub username (default $WERF_REPO_DOCKER_HUB_USERNAME)
- `--repo-github-token`

  repo GitHub token (default $WERF_REPO_GITHUB_TOKEN)
- `--repo-harbor-password`

  repo Harbor password (default $WERF_REPO_HARBOR_PASSWORD)
- `--repo-harbor-username`

  repo Harbor username (default $WERF_REPO_HARBOR_USERNAME)
- `--repo-quay-token`

  repo quay.io token (default $WERF_REPO_QUAY_TOKEN)
- `--save-build-report`

  Save build report (by default $WERF_SAVE_BUILD_REPORT or false). Its path and format configured with --build-report-path

- `--secondary-repo`

  Specify one or multiple secondary read-only repos with images that will be used as a cache.
  Also, can be specified with $WERF_SECONDARY_REPO_* (e.g. $WERF_SECONDARY_REPO_1=..., $WERF_SECONDARY_REPO_2=...)
- `--skip-image-spec-stage`

  Force skipping "imageSpec" build stage (default $WERF_SKIP_IMAGE_SPEC_STAGE or false)
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--synchronization`, `-S`

  Address of synchronizer for multiple werf processes to work with a single repo.

  Default:
  - $WERF_SYNCHRONIZATION, or
  - :local if --repo is not specified, or
  - https://delivery-sync.deckhouse.ru if --repo has been specified.

  The same address should be specified for all werf processes that work with a single repo. :local address allows execution of werf processes from a single host only
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--to`

  Destination address to copy stages to. Use archive:PATH for stage archive or [docker://]REPO for container registry.
- `--use-build-report`

  Use build report, previously saved with --save-build-report (by default $WERF_USE_BUILD_REPORT or false). Its path and format configured with --build-report-path
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit synchronization

Usage:

```
d8 delivery-kit synchronization [options]
```

Run synchronization server

Options

- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--dev`

  Enable development mode (default $WERF_DEV).
  The mode allows working with project files without doing redundant commits during debugging and development
- `--dev-branch`

  Set dev git branch name (default $WERF_DEV_BRANCH or "_werf-dev")
- `--dev-ignore`

  Add rules to ignore tracked and untracked changes in development mode (can specify multiple).
  Also, can be specified with $WERF_DEV_IGNORE_* (e.g. $WERF_DEV_IGNORE_TESTS=*_test.go, $WERF_DEV_IGNORE_DOCS=path/to/docs)
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--host`

  Bind synchronization server to the specified host (default localhost or $WERF_HOST)

- `--kube-config`

  Kubernetes config file path (default $WERF_KUBE_CONFIG, or $WERF_KUBECONFIG, or $KUBECONFIG)
- `--kube-config-base64`

  Kubernetes config data as base64 string (default $WERF_KUBE_CONFIG_BASE64 or $WERF_KUBECONFIG_BASE64 or $KUBECONFIG_BASE64)
- `--kube-context`

  Kubernetes config context (default $WERF_KUBE_CONTEXT)
- `--kubernetes`

  Use kubernetes lock-manager stages-storage-cache (default $WERF_KUBERNETES)
- `--kubernetes-namespace-prefix`

  Use specified prefix for namespaces created for lock-manager and stages-storage-cache (defaults to 'werf-synchronization-' when --kubernetes option is used or $WERF_KUBERNETES_NAMESPACE_PREFIX)
- `--local`

  Use file lock-manager and file stages-storage-cache (true by default or $WERF_LOCAL)
- `--local-lock-manager-base-dir`

  Use specified directory as base for file lock-manager (~/.werf/synchronization_server/lock_manager by default or $WERF_LOCAL_LOCK_MANAGER_BASE_DIR)
- `--local-stages-storage-cache-base-dir`

  Use specified directory as base for file stages-storage-cache (~/.werf/synchronization_server/stages_storage_cache by default or $WERF_LOCAL_STAGES_STORAGE_CACHE_BASE_DIR)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--loose-giterminism`

  Loose werf giterminism mode restrictions
- `--port`

  Bind synchronization server to the specified port (default 55581 or $WERF_PORT)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)

- `--ttl`

  Time to live for lock-manager locks and stages-storage-cache records (default $WERF_TTL)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).

d8 delivery-kit verify

Usage:

```
d8 delivery-kit verify --image-ref <image-ref> [--image-ref <image-ref>...] --verify-roots=<root-ca.pem>
[--verify-manifest] [--verify-elf-files] [options]
```

Verify images

Options

- `--container-registry-mirror`

  (Buildah-only) Use specified mirrors for docker.io
- `--debug`

  Enable debug (default $WERF_DEBUG).
- `--docker-config`

  Specify docker config directory path. Default $WERF_DOCKER_CONFIG or $DOCKER_CONFIG or ~/.docker (in the order of priority)
  Command needs granted permissions to read, pull and push images into the specified repo, to pull base images
- `--home-dir`

  Use specified dir to store werf cache files and dirs (default $WERF_HOME or ~/.werf)
- `--image-ref`

  Verify only passed references (default $WERF_IMAGE_REF separated by comma)
- `--insecure-registry`

  Use plain HTTP requests when accessing a registry (default $WERF_INSECURE_REGISTRY)
- `--log-color-mode`

  Set log color mode.
  Supported on, off and auto (based on the stdout’s file descriptor referring to a terminal) modes.
  Default $WERF_LOG_COLOR_MODE or auto mode.
- `--log-debug`

  Enable debug (default $WERF_LOG_DEBUG).
- `--log-pretty`

  Enable emojis, auto line wrapping and log process border (default $WERF_LOG_PRETTY or true).
- `--log-project-dir`

  Print current project directory path (default $WERF_LOG_PROJECT_DIR)
- `--log-quiet`

  Disable explanatory output (default $WERF_LOG_QUIET).
- `--log-terminal-width`

  Set log terminal width.
  Defaults to:
  * $WERF_LOG_TERMINAL_WIDTH
  * interactive terminal width or 140
- `--log-time`

  Add time to log entries for precise event time tracking (default $WERF_LOG_TIME or false).
- `--log-time-format`

  Specify custom log time format (default $WERF_LOG_TIME_FORMAT or RFC3339 format).
- `--log-verbose`

  Enable verbose output (default $WERF_LOG_VERBOSE).
- `--parallel`, `-p`

  Run in parallel (default $WERF_PARALLEL or true)

- `--parallel-tasks-limit`

  Parallel tasks limit, set -1 to remove the limitation (default $WERF_PARALLEL_TASKS_LIMIT or 5)
- `--quiet`

  Disable explanatory output (default $WERF_QUIET).
- `--skip-tls-verify-registry`

  Skip TLS certificate validation when accessing a registry (default $WERF_SKIP_TLS_VERIFY_REGISTRY)
- `--tmp-dir`

  Use specified dir to store tmp files and dirs (default $WERF_TMP_DIR or system tmp dir)
- `--verbose`

  Enable verbose output (default $WERF_VERBOSE).
- `--verify-bsign-elf-files`

  Enable ELF files verification for bsign signed files (default $WERF_VERIFY_BSIGN_ELF_FILES).
- `--verify-elf-files`

  Enable ELF files verification (default $WERF_VERIFY_ELF_FILES).
  When enabled, the root certificates must be specified with --verify-roots option
- `--verify-manifest`

  Enable image manifest verification (default $WERF_VERIFY_MANIFEST).
  When enabled,
  the root certificates must be specified with --verify-roots option
- `--verify-roots`

  The root certificates as path to PEM file or base64-encoded PEM (default $WERF_VERIFY_ROOTS separated by comma)

d8 delivery-kit version

Usage:

```
d8 delivery-kit version [options]
```

Print version

d8 help

Help about any command

d8 iam

Manage Deckhouse users, groups, and access grants

d8 iam access

Usage:

```
d8 iam access [options]
```

Grant or revoke access (current authz model)

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam access grant

Usage:

```
d8 iam access grant (user|group) NAME --access-level LEVEL [scope] [options]
```

Grant access to a user or group (current authz model)

Options

- `--access-level`

  Access level: User, PrivilegedUser, Editor, Admin, ClusterEditor, ClusterAdmin, SuperAdmin
- `--allow-scale`

  Allow scaling workloads
- `--dry-run`

  Print the resource(s) that would be created without applying
- `--namespace`, `-n`

  Target namespace(s) for namespaced grants (repeatable). Mutually exclusive with --scope
- `--output`, `-o`

  Output format: name|yaml|json
- `--port-forwarding`

  Allow port-forwarding
- `--scope`

  Cluster-wide scope: cluster | all-namespaces | labels=K=V[,K2=V2,...]. Mutually exclusive with -n/--namespace

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam access revoke

Usage:

```
d8 iam access revoke (user|group) NAME --access-level LEVEL [scope] [options]
```

Revoke a d8-managed access grant

Options

- `--access-level`

  Access level to revoke
- `--allow-scale`

  Match grants with allow-scale enabled
- `--dry-run`

  Resolve and print the object(s) that would be deleted without touching the cluster
- `--namespace`, `-n`

  Target namespace(s). Mutually exclusive with --scope
- `--output`, `-o`

  Output format: name|yaml|json
- `--port-forwarding`

  Match grants with port-forwarding enabled
- `--scope`

  Cluster-wide scope: cluster | all-namespaces | labels=K=V[,K2=V2,...]. Mutually exclusive with -n/--namespace

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam get

Usage:

```
d8 iam get (user|group|rule) NAME [options]
```

Show a single Deckhouse IAM object

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam get group

Alias: `groups`

Usage:

```
d8 iam get (user|group|rule) NAME group <name> [options]
```

Show detailed access for a specific group

Options

- `--output`, `-o`

  Output format: table|json|yaml

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam get rule

Alias: `rules`

Usage:

```
d8 iam get (user|group|rule) NAME rule REF [options]
```

Show a single CAR or AR with subjects, scope and reverse CR lookup

Options

- `--output`, `-o`

  Output format: text|json|yaml

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam get user

Alias: `users`

Usage:

```
d8 iam get (user|group|rule) NAME user <name> [options]
```

Show detailed access for a specific user

Options

- `--output`, `-o`

  Output format: table|json|yaml

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam group

Alias: `groups`

Usage:

```
d8 iam group [options]
```

Manage Deckhouse local groups (user-authn)

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam group add-member

Usage:

```
d8 iam group add-member GROUP [user|group] MEMBER [options]
```

Add a member (user or group) to a local group

Options

- `--create-group`

  Create the target group if it does not exist

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam group create

Usage:

```
d8 iam group create <name> [options]
```

Create a local group in Deckhouse

Options

- `--dry-run`

  Print the resource that would be created without applying
- `--output`, `-o`

  Output format: name|yaml|json

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam group delete

Usage:

```
d8 iam group delete <name> [options]
```

Delete a local group

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam group remove-member

Usage:

```
d8 iam group remove-member GROUP [user|group] MEMBER [options]
```

Remove a member (user or group) from a local group

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam list

Usage:

```
d8 iam list (users|groups|rules) [options]
```

List Deckhouse IAM objects

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam list groups

Alias: `group`

Usage:

```
d8 iam list (users|groups|rules) groups [options]
```

List all groups with their effective access

Options

- `--output`, `-o`

  Output format: table|json

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam list rules

Alias: `rule`

Usage:

```
d8 iam list (users|groups|rules) rules [options]
```

List ClusterAuthorizationRules and AuthorizationRules

Options

- `--cluster`

  Only ClusterAuthorizationRules
- `--managed-only`

  Only rules managed by d8-cli (label app.kubernetes.io/managed-by=d8-cli)
- `--manual-only`

  Only rules NOT managed by d8-cli
- `--namespace`, `-n`

  Restrict to AuthorizationRules in these namespaces (repeatable). Implicitly disables CARs unless --cluster is also set.
- `--output`, `-o`

  Output format: table|json|yaml

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam list users

Alias: `user`

Usage:

```
d8 iam list (users|groups|rules) users [options]
```

List all users with their effective access

Options

- `--output`, `-o`

  Output format: table|json

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam user

Alias: `userop`

Usage:

```
d8 iam user [options]
```

Manage Deckhouse users (user-authn)

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam user create

Usage:

```
d8 iam user create <name> --email <email> [flags] [options]
```

Create a local static user in Deckhouse

Options

- `--create-groups`

  Create groups specified by --member-of if they do not exist
- `--dry-run`

  Print the resource that would be created without applying
- `--email`

  User email address (required, must be lowercase)
- `--generate-password`

  Auto-generate a strong password (shown once on stderr)
- `--member-of`

  Add user to these groups (repeatable)
- `--output`, `-o`

  Output format: name|yaml|json
- `--password-hash`

  Use a pre-computed bcrypt hash directly (must start with $2 and be a valid bcrypt cost)
- `--password-prompt`

  Read password interactively with hidden input
- `--password-stdin`

  Read password from stdin (for CI/pipelines)
- `--ttl`

  User time-to-live (e.g. 24h, 30m). Can only be set once; expireAt will not update on change.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam user delete

Usage:

```
d8 iam user delete <name> [options]
```

Delete a local static user

Options

- `--keep-memberships`

Do not remove the user from local Group spec.members; only delete the User CR

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam user lock

Usage:

```
d8 iam user lock <username> <lockDuration> [options]
```

Lock local user in Dex for a period of time

Options

- `--timeout`

  How long to wait for completion when --wait is enabled.
- `--wait`

  Wait for UserOperation completion and print result.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam user reset-password

Alias: `resetpass`

Usage:

```
d8 iam user reset-password <username> [options]
```

Reset local user's password in Dex

Options

- `--generate-password`

  Auto-generate a strong password (shown once on stderr)
- `--password-hash`

  Use a pre-computed bcrypt hash directly (must start with $2 and be a valid bcrypt cost)
- `--password-prompt`

  Read password interactively with hidden input
- `--password-stdin`

  Read password from stdin (for CI/pipelines)
- `--timeout`

  How long to wait for completion when --wait is enabled.
- `--wait`

  Wait for UserOperation completion and print result.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam user reset2fa

Usage:

```
d8 iam user reset2fa <username> [options]
```

Reset local user's 2FA (TOTP) in Dex

Options

- `--timeout`
  How long to wait for completion when --wait is enabled.
- `--wait`
  Wait for UserOperation completion and print result.

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 iam user unlock

Usage:

```
d8 iam user unlock <username> [options]
```

Unlock local user in Dex

Options

- `--timeout`
  How long to wait for completion when --wait is enabled.
- `--wait`
  Wait for UserOperation completion and print result.

Global options

- `--context`
  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`
  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 k

Alias: `kubectl`

kubectl controls the Kubernetes cluster manager

Options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority

- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k alpha

Usage:

```
d8 k alpha [options]
```

Commands for features in alpha

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`
  The address and port of the Kubernetes API server
- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k annotate

Usage:

```
d8 k annotate [--overwrite] (-f FILENAME | TYPE NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--resource-
version=version] [options]
```

Update the annotations on a resource

Options

- `--all`

  Select all resources, in the namespace of the specified resource types.
- `--all-namespaces`, `-A`

  If true, check the specified action in all namespaces.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--field-selector`

  Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to update the annotation
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--list`

  If true, display the annotations for a given resource.
- `--local`

  If true, annotation will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--overwrite`

  If true, allow annotations to be overwritten, otherwise reject annotation updates that overwrite existing annotations.

- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--resource-version`

  If non-empty, the annotation update will only succeed if this is the current resource-version for the object. Only valid when specifying a single resource.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k api-resources

Usage:

```
d8 k api-resources [options]
```

Print the supported API resources on the server

Options

- `--api-group`

  Limit to resources in the specified API group.
- `--cached`

  Use the cached list of resources if available.
- `--categories`

  Limit to resources that belong to the specified categories.
- `--namespaced`

  If false, non-namespaced resources will be returned, otherwise returning namespaced resources by default.
- `--no-headers`

  When using the default or custom-column output format, don't print headers (default print headers).
- `--output`, `-o`

  Output format. One of: (json, yaml, name, wide).
- `--show-managed-fields`

If true, keep the managedFields when printing objects in JSON or YAML format.

- `--sort-by`

  If non-empty, sort list of resources using specified field. The field can be either 'name' or 'kind'.
- `--verbs`

  Limit to resources that support the specified verbs.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server

- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k api-versions

Usage:

```
d8 k api-versions [options]
```

Print the supported API versions on the server, in the form of "group/version"

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k apply

Usage:

```
d8 k apply (-f FILENAME | -k DIRECTORY) [options]
```

Apply a configuration to a resource by file name or stdin

Options

- `--all`

  Select all resources in the namespace of the specified resource types.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--cascade`

  Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents (e.g. Pods created by a ReplicationController). Defaults to background.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

Name of the manager used to track field ownership.

- `--filename`, `-f`

  The files that contain the configurations to apply.
- `--force`

  If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of some resources may result in inconsistency or data loss and requires confirmation.
- `--force-conflicts`

  If true, server-side apply will force the changes against conflicts.
- `--grace-period`

  Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set to 0 when --force is true (force deletion).
- `--kustomize`, `-k`

  Process a kustomization directory. This flag can't be used together with -f or -R.
- `--openapi-patch`

  If true, use openapi to calculate diff when the openapi presents and the resource can be found in the openapi spec. Otherwise, fall back to use baked-in types.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--overwrite`

  Automatically resolve conflicts between the modified and live configuration by using values from the modified configuration
- `--prune`

  Automatically delete resource objects, that do not appear in the configs and are created by either apply or create --save-config. Should be used with either -l or --all.
- `--prune-allowlist`

  Overwrite the default allowlist with for --prune
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--server-side`

  If true, apply runs in the server instead of the client.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--subresource`

  If specified, apply will operate on the subresource of the requested object. Only allowed when using --server-side.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--timeout`

  The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to

validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

- `--wait`

  If true, wait for resources to be gone before returning. This waits for finalizers.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should

contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k apply edit-last-applied

Usage:

```
d8 k apply (-f FILENAME | -k DIRECTORY) edit-last-applied (RESOURCE/NAME | -f FILENAME) [options]
```

Edit latest last-applied-configuration annotations of a resource/object

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files to use to edit the resource
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to

validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

- `--windows-line-endings`

  Defaults to the line ending native to your platform.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should

contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k apply set-last-applied

Usage:

```
d8 k apply (-f FILENAME | -k DIRECTORY) set-last-applied -f FILENAME [options]
```

Set the last-applied-configuration annotation on a live object to match the contents of a file

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--create-annotation`

  Will create 'last-applied-configuration' annotations if current objects doesn't have one
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--filename`, `-f`

  Filename, directory, or URL to files that contains the last-applied-configuration annotations
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

UID to impersonate for the operation.

- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text

log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k apply view-last-applied

Usage:

```
d8 k apply (-f FILENAME | -k DIRECTORY) view-last-applied (TYPE [NAME | -l label] | TYPE/NAME | -f
FILENAME) [options]
```

View the latest last-applied-configuration annotations of a resource/object

Options

- `--all`

  Select all resources in the namespace of the specified resource types
- `--filename`, `-f`

  Filename, directory, or URL to files that contains the last-applied-configuration annotations
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. Must be one of (yaml, json)
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k attach

Usage:

```
d8 k attach (POD | TYPE/NAME) -c CONTAINER [options]
```

Attach to a running container

Options

- `--container`, `-c`

  Container name. If omitted, use the kubectl.kubernetes.io/default-container annotation for selecting the container to be attached or the first container in the pod will be chosen
- `--pod-running-timeout`

  The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running
- `--quiet`, `-q`

  Only print output from the remote session
- `--stdin`, `-i`

Pass stdin to the container
- `--tty`, `-t`

  Stdin is a TTY

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to

contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k auth

Usage:

```
d8 k auth [options]
```

Inspect authorization

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes

- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k auth can-i

Usage:

```
d8 k auth can-i VERB [TYPE | TYPE/NAME | NONRESOURCEURL] [options]
```

Check whether an action is allowed

Options

- `--all-namespaces`, `-A`

  If true, check the specified action in all namespaces.
- `--list`

  If true, prints all allowed actions.
- `--no-headers`

  If true, prints allowed actions without headers
- `--quiet`, `-q`

  If true, suppress output and just return the exit code.
- `--subresource`

  SubResource such as pod/log or deployment/scale

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a

namespace.

- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server

- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k auth reconcile

Usage:

```
d8 k auth reconcile -f FILENAME [options]
```

Reconciles rules for RBAC role, role binding, cluster role, and cluster role binding objects

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to reconcile.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--remove-extra-permissions`

  If true, removes extra permissions added to roles
- `--remove-extra-subjects`

  If true, removes extra subjects added to rolebindings
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority

- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k auth whoami

Usage:

```
d8 k auth whoami [options]
```

Experimental: Check self subject attributes

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k autoscale

Usage:

```
d8 k autoscale (-f FILENAME | TYPE NAME | TYPE/NAME) [--min=MINPODS] --max=MAXPODS [--cpu=CPU] [--
memory=MEMORY] [options]
```

Auto-scale a deployment, replica set, stateful set, or replication controller

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--cpu`

  Target CPU utilization over all the pods. When specified as a percentage (e.g."70%" for 70% of requested CPU) it will target average utilization. When specified as quantity (e.g."500m" for 500 milliCPU) it will target average value. Value without units is treated as a quantity with miliCPU being the unit (e.g."500" is "500m").
- `--cpu-percent`

  The target average CPU utilization (represented as a percent of requested CPU) over all the pods. If it's not specified or negative, a default autoscaling policy will be used.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.

- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to autoscale.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--max`

  The upper limit for the number of pods that can be set by the autoscaler. Required.
- `--memory`

  Target memory utilization over all the pods. When specified as a percentage (e.g."60%" for 60% of requested memory) it will target average utilization. When specified as quantity (e.g."200Mi" for 200 MiB, "1Gi" for 1 GiB) it will target average value. Value without units is treated as a quantity with mebibytes being the unit (e.g."200" is "200Mi").
- `--min`

  The lower limit for the number of pods that can be set by the autoscaler. If it's not specified or negative, the server will apply a default value.
- `--name`

  The name for the newly created object. If not specified, the name of the input resource will be used.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS

- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k certificate

Usage:

```
d8 k certificate SUBCOMMAND [options]
```

Modify certificate resources

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

The name of the kubeconfig user to use

- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k certificate approve

Usage:

```
d8 k certificate SUBCOMMAND approve (-f FILENAME | NAME) [options]
```

Approve a certificate signing request

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to update
- `--force`

  Update the CSR even if it is already approved.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS

- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k certificate deny

Usage:

```
d8 k certificate SUBCOMMAND deny (-f FILENAME | NAME) [options]
```

Deny a certificate signing request

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to update
- `--force`

  Update the CSR even if it is already denied.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k cluster-info

Usage:

```
d8 k cluster-info [options]
```

Display cluster information

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory

- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k cluster-info dump

Usage:

```
d8 k cluster-info dump [options]
```

Dump relevant information for debugging and diagnosis

Options

- `--all-namespaces`, `-A`

  If true, dump all namespaces. If true, --namespaces is ignored.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--namespaces`

  A comma separated list of namespaces to dump.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--output-directory`

  Where to output the files. If empty or '-' uses stdout, otherwise creates a directory hierarchy in that directory
- `--pod-running-timeout`

  The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k completion

Usage:

```
d8 k completion SHELL [options]
```

Output shell completion code for the specified shell (bash, zsh, fish, or powershell)

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.

- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config

Usage:

```
d8 k config SUBCOMMAND [options]
```

Modify kubeconfig files

Options

- `--kubeconfig`

  use a particular kubeconfig file

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to

- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config current-context

Usage:

```
d8 k config SUBCOMMAND current-context [options]
```

Display the current-context

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config delete-cluster

Usage:

```
d8 k config SUBCOMMAND delete-cluster NAME [options]
```

Delete the specified cluster from the kubeconfig

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.

- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config delete-context

Usage:

```
d8 k config SUBCOMMAND delete-context NAME [options]
```

Delete the specified context from the kubeconfig

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should

contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config delete-user

Usage:

```
d8 k config SUBCOMMAND delete-user NAME [options]
```

Delete the specified user from the kubeconfig

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file

- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config get-clusters

Usage:

```
d8 k config SUBCOMMAND get-clusters [options]
```

Display clusters defined in the kubeconfig

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory

- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config get-contexts

Usage:

```
d8 k config SUBCOMMAND get-contexts [(-o|--output=)name)] [options]
```

Describe one or many contexts

Options

- `--no-headers`
  When using the default or custom-column output format, don't print headers (default print headers).
- `--output`, `-o`
  Output format. One of: (name).

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  use a particular kubeconfig file
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config get-users

Usage:

```
d8 k config SUBCOMMAND get-users [options]
```

Display users defined in the kubeconfig

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config rename-context

Usage:

```
d8 k config SUBCOMMAND rename-context CONTEXT_NAME NEW_NAME [options]
```

Rename a context from the kubeconfig file

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config set

Usage:

```
d8 k config SUBCOMMAND set PROPERTY_NAME PROPERTY_VALUE [options]
```

Set an individual value in a kubeconfig file

Options

- `--set-raw-bytes`

  When writing a []byte PROPERTY_VALUE, write the given string directly without base64 decoding.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server

- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config set-cluster

Usage:

```
d8 k config SUBCOMMAND set-cluster NAME [--server=server] [--certificate-
authority=path/to/certificate/authority] [--insecure-skip-tls-verify=true] [--tls-server-
name=example.com] [options]
```

Set a cluster entry in kubeconfig

Options

- `--certificate-authority`

  Path to certificate-authority file for the cluster entry in kubeconfig
- `--embed-certs`

  embed-certs for the cluster entry in kubeconfig
- `--insecure-skip-tls-verify`

  insecure-skip-tls-verify for the cluster entry in kubeconfig
- `--proxy-url`

  proxy-url for the cluster entry in kubeconfig
- `--server`

  server for the cluster entry in kubeconfig
- `--tls-server-name`

  tls-server-name for the cluster entry in kubeconfig

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config set-context

Usage:

```
d8 k config SUBCOMMAND set-context [NAME | --current] [--cluster=cluster_nickname] [--user=user_nickname]
[--namespace=namespace] [options]
```

Set a context entry in kubeconfig

Options

- `--cluster`
  cluster for the context entry in kubeconfig
- `--current`
  Modify the current context
- `--namespace`
  namespace for the context entry in kubeconfig
- `--user`
  user for the context entry in kubeconfig

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  use a particular kubeconfig file
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config set-credentials

Usage:

```
d8 k config SUBCOMMAND set-credentials NAME [--client-certificate=path/to/certfile] [--client-key=path/to/keyfile] [--token=bearer_token] [--username=basic_user] [--password=basic_password] [--auth-provider=provider_name] [--auth-provider-arg=key=value] [--exec-command=exec_command] [--exec-api-version=exec_api_version] [--exec-arg=arg] [--exec-env=key=value] [options]
```

Set a user entry in kubeconfig

Options

- `--auth-provider`

  Auth provider for the user entry in kubeconfig
- `--auth-provider-arg`

  'key=value' arguments for the auth provider
- `--client-certificate`

  Path to client-certificate file for the user entry in kubeconfig
- `--client-key`

  Path to client-key file for the user entry in kubeconfig
- `--embed-certs`

  Embed client cert/key for the user entry in kubeconfig
- `--exec-api-version`

  API version of the exec credential plugin for the user entry in kubeconfig
- `--exec-arg`

  New arguments for the exec credential plugin command for the user entry in kubeconfig
- `--exec-command`

  Command for the exec credential plugin for the user entry in kubeconfig
- `--exec-env`

  'key=value' environment values for the exec credential plugin
- `--exec-interactive-mode`

  InteractiveMode of the exec credentials plugin for the user entry in kubeconfig
- `--exec-provide-cluster-info`

  ProvideClusterInfo of the exec credentials plugin for the user entry in kubeconfig
- `--password`

  password for the user entry in kubeconfig
- `--token`

  token for the user entry in kubeconfig
- `--username`

  username for the user entry in kubeconfig

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--user`

  The name of the kubeconfig user to use
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config unset

Usage:

```
d8 k config SUBCOMMAND unset PROPERTY_NAME [options]
```

Unset an individual value in a kubeconfig file

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  use a particular kubeconfig file
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`
  The address and port of the Kubernetes API server

- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config use-context

Alias: `use`

Usage:

```
d8 k config SUBCOMMAND use-context CONTEXT_NAME [options]
```

Set the current-context in a kubeconfig file

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting

KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k config view

Usage:

```
d8 k config SUBCOMMAND view [options]
```

Display merged kubeconfig settings or a specified kubeconfig file

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--flatten`

  Flatten the resulting kubeconfig file into self-contained output (useful for creating portable kubeconfig files)
- `--merge`

  Merge the full hierarchy of kubeconfig files
- `--minify`

  Remove all information not used by current-context from the output
- `--output`, `-o`

Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

- `--raw`

  Display raw byte data and sensitive data
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  use a particular kubeconfig file
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k cordon

Usage:

```
d8 k cordon NODE [options]
```

Mark node as unschedulable

Options

- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k cp

Usage:

```
d8 k cp <file-spec-src> <file-spec-dest> [options]
```

Copy files and directories to and from containers

Options

- `--container`, `-c`

  Container name. If omitted, use the kubectl.kubernetes.io/default-container annotation for selecting the container to be attached or the first container in the pod will be chosen
- `--no-preserve`

  The copied file/directory's ownership and permissions will not be preserved in the container
- `--retries`

  Set number of retries to complete a copy operation from a container. Specify 0 to disable or any negative value for infinite retrying. The default is 0 (no retry).

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create

Usage:

```
d8 k create -f FILENAME [options]
```

Create a resource from a file or from stdin

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--edit`

  Edit the API resource before creating
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files to use to create the resource
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--raw`

  Raw URI to POST to the server. Uses the transport specified by the kubeconfig file.
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related

manifests organized within the same directory.

- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--windows-line-endings`

  Only relevant if --edit=true. Defaults to the line ending native to your platform.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting

KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create clusterrole

Usage:

```
d8 k create -f FILENAME clusterrole NAME --verb=verb --resource=resource.group [--resource-
name=resourcename] [--dry-run=server|client|none] [options]
```

Create a cluster role

Options

- `--aggregation-rule`

  An aggregation label selector for combining ClusterRoles.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.

- `--non-resource-url`

  A partial url that user should have access to.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--resource`

  Resource that the rule applies to
- `--resource-name`

  Resource in the white list that the rule applies to, repeat this flag for multiple items
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--verb`

  Verb that applies to the resources contained in the rule

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS

connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create clusterrolebinding

Usage:

```
d8 k create -f FILENAME clusterrolebinding NAME --clusterrole=NAME [--user=username] [--group=groupname]
[--serviceaccount=namespace:serviceaccountname] [--dry-run=server|client|none] [options]
```

Create a cluster role binding for a particular cluster role

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--clusterrole`

  ClusterRole this ClusterRoleBinding should reference
- `--dry-run`

Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--field-manager`

  Name of the manager used to track field ownership.
- `--group`

  Groups to bind to the clusterrole. The flag can be repeated to add multiple groups.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--serviceaccount`

  Service accounts to bind to the clusterrole, in the format :. The flag can be repeated to add multiple service accounts.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--user`

  Usernames to bind to the clusterrole. The flag can be repeated to add multiple users.
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create configmap

Alias: `cm`

Usage:

```
d8 k create -f FILENAME configmap NAME [--from-file=[key=]source] [--from-literal=key1=value1] [--dry-run=server|client|none] [options]
```

Create a config map from a local file, directory or literal value

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.

- `--append-hash`

  Append a hash of the configmap to its name.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--from-env-file`

  Specify the path to a file to read lines of key=val pairs to create a configmap.
- `--from-file`

  Key file can be specified using its file path, in which case file basename will be used as configmap key, or optionally with a key and file path, in which case the given key will be used. Specifying a directory will iterate each named file in the directory whose basename is a valid configmap key.
- `--from-literal`

  Specify a key and literal value to insert in configmap (i.e. mykey=somevalue)
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create cronjob

Alias: `cj`

Usage:

```
d8 k create -f FILENAME cronjob NAME --image=image --schedule='0/5 * * * ?' -- [COMMAND] [args...]
[options]
```

Create a cron job with the specified name

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--image`

  Image name to run.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--restart`

  job's restart policy. supported values: OnFailure, Never
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--schedule`

  A schedule in the Cron format the job should be run with.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS

- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create deployment

Alias: `deploy`

Usage:

```
d8 k create -f FILENAME deployment NAME --image=image -- [COMMAND] [args...] [options]
```

Create a deployment with the specified name

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--image`

  Image names to run. A deployment can have multiple images set for multi-container pod.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--port`

  The containerPort that this deployment exposes.
- `--replicas`, `-r`

  Number of replicas to create. Default is 1.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS

- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create ingress

Alias: `ing`

Usage:

```
d8 k create -f FILENAME ingress NAME --rule=host/path=service:port[,tls[=secret]]  [options]
```

Create an ingress with the specified name

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--annotation`

  Annotation to insert in the ingress object, in the format annotation=value
- `--class`

  Ingress Class to be used
- `--default-backend`

  Default service for backend, in format of svcname:port
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--rule`

  Rule in format host/path=service:port[,tls=secretname]. Paths containing the leading character '*' are considered pathType=Prefix. tls argument is optional.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create job

Usage:

```
d8 k create -f FILENAME job NAME --image=image [--from=cronjob/name] -- [COMMAND] [args...] [options]
```

Create a job with the specified name

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--from`

  The name of the resource to create a Job from (only cronjob is supported).
- `--image`

  Image name to run.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create namespace

Alias: `ns`

Usage:

```
d8 k create -f FILENAME namespace NAME [--dry-run=server|client|none] [options]
```

Create a namespace with the specified name

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create poddisruptionbudget

Alias: `pdb`

Usage:

```
d8 k create -f FILENAME poddisruptionbudget NAME --selector=SELECTOR --min-available=N [--dry-
run=server|client|none] [options]
```

Create a pod disruption budget with the specified name

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--max-unavailable`

  The maximum number or percentage of unavailable pods this budget requires.
- `--min-available`

  The minimum number or percentage of available pods this budget requires.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--selector`

  A label selector to use for this budget. Only equality-based selector requirements are supported.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create priorityclass

Alias: `pc`

Usage:

```
d8 k create -f FILENAME priorityclass NAME --value=VALUE --global-default=BOOL [--dry-
run=server|client|none] [options]
```

Create a priority class with the specified name

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--description`

  description is an arbitrary string that usually provides guidelines on when this priority class should be used.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--global-default`

  global-default specifies whether this PriorityClass should be considered as the default priority.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--preemption-policy`

  preemption-policy is the policy for preempting pods with lower priority.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--value`

  the value of this priority class.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create quota

Alias: `resourcequota`

Usage:

```
d8 k create -f FILENAME quota NAME [--hard=key1=value1,key2=value2] [--scopes=Scope1,Scope2] [--dry-
run=server|client|none] [options]
```

Create a quota with the specified name

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--hard`

  A comma-delimited set of resource=quantity pairs that define a hard limit.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--scopes`

  A comma-delimited set of quota scopes that must all match each object tracked by the quota.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS

- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create role

Usage:

```
d8 k create -f FILENAME role NAME --verb=verb --resource=resource.group/subresource [--resource-
name=resourcename] [--dry-run=server|client|none] [options]
```

Create a role with single rule

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--resource`

  Resource that the rule applies to
- `--resource-name`

  Resource in the white list that the rule applies to, repeat this flag for multiple items
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--verb`

  Verb that applies to the resources contained in the rule

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create rolebinding

Usage:

```
d8 k create -f FILENAME rolebinding NAME --clusterrole=NAME|--role=NAME [--user=username] [--
group=groupname] [--serviceaccount=namespace:serviceaccountname] [--dry-run=server|client|none] [options]
```

Create a role binding for a particular role or cluster role

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--clusterrole`

  ClusterRole this RoleBinding should reference
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--group`

  Groups to bind to the role. The flag can be repeated to add multiple groups.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--role`

  Role this RoleBinding should reference
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--serviceaccount`

  Service accounts to bind to the role, in the format :. The flag can be repeated to add multiple service accounts.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--user`

  Usernames to bind to the role. The flag can be repeated to add multiple users.
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.

- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create secret

Usage:

```
d8 k create -f FILENAME secret (docker-registry | generic | tls) [options]
```

Create a secret using a specified subcommand

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`
  The address and port of the Kubernetes API server

- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create secret docker-registry

Usage:

```
d8 k create -f FILENAME secret (docker-registry | generic | tls) docker-registry NAME --docker-
username=user --docker-password=password --docker-email=email [--docker-server=string] [--from-file=
[key=]source] [--dry-run=server|client|none] [options]
```

Create a secret for use with a Docker registry

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--append-hash`

  Append a hash of the secret to its name.
- `--docker-email`

  Email for Docker registry
- `--docker-password`

  Password for Docker registry authentication
- `--docker-server`

  Server location for Docker registry
- `--docker-username`

  Username for Docker registry authentication
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--from-file`

  Key files can be specified using their file path, in which case a default name of .dockerconfigjson will be given to them, or optionally with a name and file path, in which case the given name will be used. Specifying a directory will iterate each named file in the directory that is a valid secret key. For this command, the key should always be .dockerconfigjson.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the

annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

Password for basic authentication to the API server

- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create secret generic

Usage:

```
d8 k create -f FILENAME secret (docker-registry | generic | tls) generic NAME [--type=string] [--from-file=[key=]source] [--from-literal=key1=value1] [--dry-run=server|client|none] [options]
```

Create a secret from a local file, directory, or literal value

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--append-hash`

  Append a hash of the secret to its name.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--from-env-file`

  Specify the path to a file to read lines of key=val pairs to create a secret.
- `--from-file`

  Key files can be specified using their file path, in which case a default name will be given to them, or optionally with a name and file path, in which case the given name will be used. Specifying a directory will iterate each named file in the directory that is a valid secret key.
- `--from-literal`

  Specify a key and literal value to insert in secret (i.e. mykey=somevalue)

- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--type`

  The type of secret to create
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create secret tls

Usage:

```
d8 k create -f FILENAME secret (docker-registry | generic | tls) tls NAME --cert=path/to/cert/file --
key=path/to/key/file [--dry-run=server|client|none] [options]
```

Create a TLS secret

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--append-hash`

  Append a hash of the secret to its name.
- `--cert`

  Path to PEM encoded public key certificate.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

Name of the manager used to track field ownership.

- `--key`

  Path to private key associated with given certificate.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting

KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service

Alias: `svc`

Usage:

```
d8 k create -f FILENAME service [options]
```

Create a service using a specified subcommand

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

Path to a cert file for the certificate authority

- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service clusterip

Usage:

```
d8 k create -f FILENAME service clusterip NAME [--tcp=<port>:<targetPort>] [--dry-run=server|client|none]
[options]
```

Create a ClusterIP service

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--clusterip`

  Assign your own ClusterIP or set to 'None' for a 'headless' service (no loadbalancing).
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--tcp`

  Port pairs can be specified as ':'.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority

- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service externalname

Usage:

```
d8 k create -f FILENAME service externalname NAME --external-name external.name [--dry-
run=server|client|none] [options]
```

Create an ExternalName service

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--external-name`

  External name of service
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--tcp`

  Port pairs can be specified as ':'.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS

- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service loadbalancer

Usage:

```
d8 k create -f FILENAME service loadbalancer NAME [--tcp=port:targetPort] [--dry-run=server|client|none]
[options]
```

Create a LoadBalancer service

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--tcp`

  Port pairs can be specified as ':'.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create service nodeport

Usage:

```
d8 k create -f FILENAME service nodeport NAME [--tcp=port:targetPort] [--dry-run=server|client|none]
[options]
```

Create a NodePort service

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--node-port`

  Port used to expose the service on each node in a cluster.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--tcp`

  Port pairs can be specified as ':'.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create serviceaccount

Alias: `sa`

Usage:

```
d8 k create -f FILENAME serviceaccount NAME [--dry-run=server|client|none] [options]
```

Create a service account with the specified name

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS

connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k create token

Usage:

```
d8 k create -f FILENAME token SERVICE_ACCOUNT_NAME [options]
```

Request a service account token

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--audience`

  Audience of the requested token. If unset, defaults to requesting a token for use with the Kubernetes API server. May be repeated to request a token valid for multiple audiences.

- `--bound-object-kind`

  Kind of an object to bind the token to. Supported kinds are Node, Pod, Secret. If set, --bound-object-name must be provided.
- `--bound-object-name`

  Name of an object to bind the token to. The token will expire when the object is deleted. Requires --bound-object-kind.
- `--bound-object-uid`

  UID of an object to bind the token to. Requires --bound-object-kind and --bound-object-name. If unset, the UID of the existing object is used.
- `--duration`

  Requested lifetime of the issued token. If not set or if set to 0, the lifetime will be determined by the server automatically. The server may return a token with a longer or shorter lifetime.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes

- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k debug

Usage:

```
d8 k debug (POD | TYPE[[.VERSION].GROUP]/NAME) [ -- COMMAND [args...] ] [options]
```

Create debugging sessions for troubleshooting workloads and nodes

Options

- `--arguments-only`

  If specified, everything after -- will be passed to the new container as Args instead of Command.
- `--attach`

  If true, wait for the container to start running, and then attach as if 'kubectl attach ...' were called. Default false, unless '-i/--stdin' is set, in which case the default is true.
- `--container`, `-c`

  Container name to use for debug container.
- `--copy-to`

  Create a copy of the target Pod with this name.
- `--custom`

  Path to a JSON or YAML file containing a partial container spec to customize built-in debug profiles.
- `--env`

  Environment variables to set in the container.
- `--filename`, `-f`

identifying the resource to debug
- `--image`

  Container image to use for debug container. If not specified, the platform's recommended image will be used.
- `--image-pull-policy`

  The image pull policy for the container. If left empty, this value will not be specified by the client and defaulted by the server.
- `--keep-annotations`

  If true, keep the original pod annotations.(This flag only works when used with '--copy-to')
- `--keep-init-containers`

  Run the init containers for the pod. Defaults to true.(This flag only works when used with '--copy-to')
- `--keep-labels`

  If true, keep the original pod labels.(This flag only works when used with '--copy-to')
- `--keep-liveness`

  If true, keep the original pod liveness probes.(This flag only works when used with '--copy-to')
- `--keep-readiness`

  If true, keep the original pod readiness probes.(This flag only works when used with '--copy-to')
- `--keep-startup`

  If true, keep the original startup probes.(This flag only works when used with '--copy-to')
- `--profile`

  Options are "legacy", "general", "baseline", "netadmin", "restricted" or "sysadmin".
- `--quiet`, `-q`

  If true, suppress informational messages.
- `--replace`

  When used with '--copy-to', delete the original Pod.
- `--same-node`

  When used with '--copy-to', schedule the copy of target Pod on the same node.
- `--set-image`

  When used with '--copy-to', a list of name=image pairs for changing container images, similar to how 'kubectl set image' works.
- `--share-processes`

  When used with '--copy-to', enable process namespace sharing in the copy.
- `--stdin`, `-i`

  Keep stdin open on the container(s) in the pod, even if nothing is attached.
- `--target`

  When using an ephemeral container, target processes in this container name.
- `--tty`, `-t`

  Allocate a TTY for the debugging container.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k delete

Usage:

```
d8 k delete ([-f FILENAME] | [-k DIRECTORY] | TYPE [(NAME | -l label | --all)]) [options]
```

Delete resources by file names, stdin, resources and names, or by resources and label selector

Options

- `--all`

  Delete all resources, in the namespace of the specified resource types.
- `--all-namespaces`, `-A`

  If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.
- `--cascade`

  Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents (e.g. Pods created by a ReplicationController). Defaults to background.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-selector`

  Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.
- `--filename`, `-f`

  containing the resource to delete.
- `--force`

  If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of some resources may result in inconsistency or data loss and requires confirmation.
- `--grace-period`

  Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set to 0 when --force is true (force deletion).
- `--ignore-not-found`

  Treat "resource not found" as a successful delete. Defaults to "true" when --all is specified.
- `--interactive`, `-i`

  If true, delete resource only when user confirms.
- `--kustomize`, `-k`

  Process a kustomization directory. This flag can't be used together with -f or -R.
- `--now`

  If true, resources are signaled for immediate shutdown (same as --grace-period=1).
- `--output`, `-o`

  Output mode. Use "-o name" for shorter output (resource/name).
- `--raw`

  Raw URI to DELETE to the server. Uses the transport specified by the kubeconfig file.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--timeout`

  The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object
- `--wait`

  If true, wait for resources to be gone before returning. This waits for finalizers.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k describe

Usage:

```
d8 k describe (-f FILENAME | TYPE [NAME_PREFIX | -l label] | TYPE/NAME) [options]
```

Show details of a specific resource or group of resources

Options

- `--all-namespaces`, `-A`

  If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.
- `--chunk-size`

  Return large lists in chunks rather than all at once. Pass 0 to disable. This flag is beta and may change in the future.
- `--filename`, `-f`

  Filename, directory, or URL to files containing the resource to describe
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-events`

  If true, display events related to the described object.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k diff

Usage:

```
d8 k diff -f FILENAME [options]
```

Diff the live version against a would-be applied version

Options

- `--concurrency`

Number of objects to process in parallel when diffing against the live version. Larger number = faster, but more memory, I/O and CPU over that shorter period of time.

- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files contains the configuration to diff
- `--force-conflicts`

  If true, server-side apply will force the changes against conflicts.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--prune`

  Include resources that would be deleted by pruning. Can be used with -l and default shows all resources would be pruned
- `--prune-allowlist`

  Overwrite the default allowlist with for --prune
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--server-side`

  If true, apply runs in the server instead of the client.
- `--show-managed-fields`

  If true, include managed fields in the diff.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k drain

Usage:

```
d8 k drain NODE [options]
```

Drain node in preparation for maintenance

Options

- `--chunk-size`

  Return large lists in chunks rather than all at once. Pass 0 to disable. This flag is beta and may change in the future.
- `--delete-emptydir-data`

  Continue even if there are pods using emptyDir (local data that will be deleted when the node is drained).
- `--disable-eviction`

  Force drain to use delete, even if eviction is supported. This will bypass checking

PodDisruptionBudgets, use with caution.

- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--force`

  Continue even if there are pods that do not declare a controller.
- `--grace-period`

  Period of time in seconds given to each pod to terminate gracefully. If negative, the default value specified in the pod will be used.
- `--ignore-daemonsets`

  Ignore DaemonSet-managed pods.
- `--pod-selector`

  Label selector to filter pods on the node
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--skip-wait-for-delete-timeout`

  If pod DeletionTimestamp older than N seconds, skip waiting for the pod. Seconds must be greater than 0 to skip.
- `--timeout`

  The length of time to wait before giving up, zero means infinite

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k edit

Usage:

```
d8 k edit (RESOURCE/NAME | -f FILENAME) [options]
```

Edit a resource on the server

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files to use to edit the resource
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--output-patch`

Output the patch if the resource is edited.

- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--subresource`

  If specified, edit will operate on the subresource of the requested object.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--windows-line-endings`

  Defaults to the line ending native to your platform.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k events

Usage:

```
d8 k events [(-o|--output=)json|yaml|name|go-template|go-template-
file|template|templatefile|jsonpath|jsonpath-as-json|jsonpath-file] [--for TYPE/NAME] [--watch] [--
types=Normal,Warning] [options]
```

List events

Options

- `--all-namespaces`, `-A`

  If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.
- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.

- `--chunk-size`

  Return large lists in chunks rather than all at once. Pass 0 to disable. This flag is beta and may change in the future.
- `--for`

  Filter events to only those pertaining to the specified resource.
- `--no-headers`

  When using the default output format, don't print headers.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--types`

  Output only events of given types.
- `--watch`, `-w`

  After listing the requested events, watch for more events.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k exec

Usage:

```
d8 k exec (POD | TYPE/NAME) [-c CONTAINER] [flags] -- COMMAND [args...] [options]
```

Execute a command in a container

Options

- `--container`, `-c`

  Container name. If omitted, use the kubectl.kubernetes.io/default-container annotation for selecting the container to be attached or the first container in the pod will be chosen
- `--filename`, `-f`

  to use to exec into the resource
- `--pod-running-timeout`

  The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running
- `--quiet`, `-q`

  Only print output from the remote session
- `--stdin`, `-i`

  Pass stdin to the container
- `--tty`, `-t`

  Stdin is a TTY

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k explain

Usage:

```
d8 k explain TYPE [--recursive=FALSE|TRUE] [--api-version=api-version-group] [-o|--
output=plaintext|plaintext-openapiv2] [options]
```

Get documentation for a resource

Options

- `--api-version`

  Get different explanations for particular API version (API group/version)
- `--output`, `-o`

  Format in which to render the schema (plaintext, plaintext-openapiv2)
- `--recursive`

  Print the fields of fields (Currently only 1 level deep)

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k expose

Usage:

```
d8 k expose (-f FILENAME | TYPE NAME) [--port=port] [--protocol=TCP|UDP|SCTP] [--target-port=number-or-name] [--name=name] [--external-ip=external-ip-of-service] [--type=type] [options]
```

Take a replication controller, service, deployment or pod and expose it as a new Kubernetes service

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--cluster-ip`

  ClusterIP to be assigned to the service. Leave empty to auto-allocate, or set to 'None' to create a headless service.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent,

without sending it. If server strategy, submit server-side request without persisting the resource.

- `--external-ip`

  Additional external IP address (not managed by Kubernetes) to accept for the service. If this IP is routed to a node, the service can be accessed by this IP in addition to its generated service IP.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to expose a service
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--labels`, `-l`

  Labels to apply to the service created by this call.
- `--load-balancer-ip`

  IP to assign to the LoadBalancer. If empty, an ephemeral IP will be created and used (cloud-provider specific).
- `--name`

  The name for the newly created object.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--override-type`

  The method used to override the generated object: json, merge, or strategic.
- `--overrides`

  An inline JSON override for the generated object. If this is non-empty, it is used to override the generated object. Requires that the object supply a valid apiVersion field.
- `--port`

  The port that the service should serve on. Copied from the resource being exposed, if unspecified
- `--protocol`

  The network protocol for the service to be created. Default is 'TCP'.
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--selector`

  A label selector to use for this service. Only equality-based selector requirements are supported. If empty (the default) infer the selector from the replication controller or replica set.)
- `--session-affinity`

  If non-empty, set the session affinity for the service to this; legal values: 'None', 'ClientIP'
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--target-port`

  Name or number for the port on the container that the service should direct traffic to. Optional.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--type`

Type for this service: ClusterIP, NodePort, LoadBalancer, or ExternalName. Default is 'ClusterIP'.

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`
  The address and port of the Kubernetes API server
- `--tls-server-name`
  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

Bearer token for authentication to the API server

- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k get

Usage:

```
d8 k get [(-o|--output=)json|yaml|name|go-template|go-template-
file|template|templatefile|jsonpath|jsonpath-as-json|jsonpath-file|custom-columns|custom-columns-
file|wide] (TYPE[.VERSION][.GROUP] [NAME | -l label] | TYPE[.VERSION][.GROUP]/NAME ...) [flags] [options]
```

Display one or many resources

Options

- `--all-namespaces`, `-A`

  If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--chunk-size`

  Return large lists in chunks rather than all at once. Pass 0 to disable. This flag is beta and may change in the future.
- `--field-selector`

  Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--ignore-not-found`

  If set to true, suppresses NotFound error for specific objects that do not exist. Using this flag with commands that query for collections of resources has no effect when no resources are found.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--label-columns`, `-L`

  Accepts a comma separated list of labels that are going to be presented as columns. Names are case-sensitive. You can also use multiple flag options like -L label1 -L label2...
- `--no-headers`

  When using the default or custom-column output format, don't print headers (default print headers).
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file, custom-columns, custom-columns-file, wide). See custom columns [https://kubernetes.io/docs/reference/kubectl/#custom-columns], golang template [http://golang.org/pkg/text/template/#pkg-overview] and jsonpath template [https://kubernetes.io/docs/reference/kubectl/jsonpath/].
- `--output-watch-events`

Output watch event objects when --watch or --watch-only is used. Existing objects are output as initial ADDED events.

- `--raw`

  Raw URI to request from the server. Uses the transport specified by the kubeconfig file.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--server-print`

  If true, have the server return the appropriate table output. Supports extension APIs and CRDs.
- `--show-kind`

  If present, list the resource type for the requested object(s).
- `--show-labels`

  When printing, show all labels as the last column (default hide labels column)
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--sort-by`

  If non-empty, sort list types using this field specification. The field specification is expressed as a JSONPath expression (e.g. '{.metadata.name}'). The field in the API resource specified by this JSONPath expression must be an integer or a string.
- `--subresource`

  If specified, gets the subresource of the requested object.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--watch`, `-w`

  After listing/getting the requested object, watch for changes.
- `--watch-only`

  Watch for changes to the requested object(s), without listing/getting first.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k kustomize

Usage:

```
d8 k kustomize DIR [options]
```

Build a kustomization target from a directory or URL

Options

- `--as-current-user`

  use the uid and gid of the command executor to run the function in the container
- `--enable-alpha-plugins`

enable kustomize plugins
- `--enable-helm`

  Enable use of the Helm chart inflator generator.
- `--enable-managedby-label`

  enable adding app.kubernetes.io/managed-by
- `--env`, `-e`

  a list of environment variables to be used by functions
- `--helm-api-versions`

  Kubernetes api versions used by Helm for Capabilities.APIVersions
- `--helm-command`

  helm command (path to executable)
- `--helm-debug`

  Enable debug output from the Helm chart inflator generator.
- `--helm-kube-version`

  Kubernetes version used by Helm for Capabilities.KubeVersion
- `--load-restrictor`

  if set to 'LoadRestrictionsNone', local kustomizations may load files from outside their root. This does, however, break the relocatability of the kustomization.
- `--mount`

  a list of storage options read from the filesystem
- `--network`

  enable network access for functions that declare it
- `--network-name`

  the docker network to run the container in
- `--output`, `-o`

  If specified, write output to this path.
- `--reorder`

  Reorder the resources just before output. Use 'legacy' to apply a legacy reordering (Namespaces first, Webhooks last, etc). Use 'none' to suppress a final reordering.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k label

Usage:

```
d8 k label [--overwrite] (-f FILENAME | TYPE NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--resource-
version=version] [options]
```

Update the labels on a resource

Options

- `--all`

  Select all resources, in the namespace of the specified resource types
- `--all-namespaces`, `-A`

  If true, check the specified action in all namespaces.

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--field-selector`

  Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to update the labels
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--list`

  If true, display the labels for a given resource.
- `--local`

  If true, label will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--overwrite`

  If true, allow labels to be overwritten, otherwise reject label updates that overwrite existing labels.
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--resource-version`

  If non-empty, the labels update will only succeed if this is the current resource-version for the object. Only valid when specifying a single resource.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k logs

Usage:

```
d8 k logs [-f] [-p] (POD | TYPE/NAME) [-c CONTAINER] [options]
```

Print the logs for a container in a pod

Options

- `--all-containers`
  Get all containers' logs in the pod(s).
- `--all-pods`
  Get logs from all pod(s). Sets prefix to true.
- `--container`, `-c`
  Print the logs of this container
- `--follow`, `-f`
  Specify if the logs should be streamed.
- `--ignore-errors`
  If watching / following pod logs, allow for any errors that occur to be non-fatal
- `--insecure-skip-tls-verify-backend`
  Skip verifying the identity of the kubelet that logs are requested from. In theory, an attacker could provide invalid log content back. You might want to use this if your kubelet serving certificates have expired.
- `--limit-bytes`
  Maximum bytes of logs to return. Defaults to no limit.
- `--max-log-requests`
  Specify maximum number of concurrent logs to follow when using by a selector. Defaults to 5.
- `--pod-running-timeout`
  The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running
- `--prefix`
  Prefix each log line with the log source (pod name and container name)
- `--previous`, `-p`
  If true, print the logs for the previous instance of the container in a pod if it exists.
- `--selector`, `-l`
  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--since`
  Only return logs newer than a relative duration like 5s, 2m, or 3h. Defaults to all logs. Only one of since-time / since may be used.
- `--since-time`
  Only return logs after a specific date (RFC3339). Defaults to all logs. Only one of since-time / since may be used.
- `--tail`
  Lines of recent log file to display. Defaults to -1 with no selector, showing all log lines otherwise 10, if a selector is provided.
- `--timestamps`
  Include timestamps on each line in the log output

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k options

Usage:

```
d8 k options [options]
```

Print the list of flags inherited by all commands

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)

- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k patch

Usage:

```
d8 k patch (-f FILENAME | TYPE NAME) [-p PATCH|--patch-file FILE] [options]
```

Update fields of a resource

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to update
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--local`

  If true, patch will operate on the content of the file, not the server-side resource.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--patch`, `-p`

  The patch to be applied to the resource JSON file.
- `--patch-file`

  A file containing a patch to be applied to the resource.
- `--record`

Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.

- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--subresource`

  If specified, patch will operate on the subresource of the requested object.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--type`

  The type of patch being provided; one of [json merge strategic]

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k plugin

Usage:

```
d8 k plugin [flags] [options]
```

Provides utilities for interacting with plugins

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use

- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k plugin list

Usage:

```
d8 k plugin [flags] list [options]
```

List all visible plugin executables on a user's PATH

Options

- `--name-only`

  If true, display only the binary name of each plugin, rather than its full path

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k port-forward

Usage:

```
d8 k port-forward TYPE/NAME [options] [LOCAL_PORT:]REMOTE_PORT [...[LOCAL_PORT_N:]REMOTE_PORT_N]
```

Forward one or more local ports to a pod

Options

- `--address`

  Addresses to listen on (comma separated). Only accepts IP addresses or localhost as a value. When localhost is supplied, kubectl will try to bind on both 127.0.0.1 and ::1 and will fail if neither of these addresses are available to bind.
- `--pod-running-timeout`

  The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k proxy

Usage:

```
d8 k proxy [--port=PORT] [--www=static-dir] [--www-prefix=prefix] [--api-prefix=prefix] [options]
```

Run a proxy to the Kubernetes API server

Options

- `--accept-hosts`

  Regular expression for hosts that the proxy should accept.
- `--accept-paths`

  Regular expression for paths that the proxy should accept.
- `--address`

  The IP address on which to serve on.
- `--api-prefix`

  Prefix to serve the proxied API under.
- `--append-server-path`

  If true, enables automatic path appending of the kube context server path to each request.

- `--disable-filter`

  If true, disable request filtering in the proxy. This is dangerous, and can leave you vulnerable to XSRF attacks, when used with an accessible port.
- `--keepalive`

  keepalive specifies the keep-alive period for an active network connection. Set to 0 to disable keepalive.
- `--port`, `-p`

  The port on which to run the proxy. Set to 0 to pick a random port.
- `--reject-methods`

  Regular expression for HTTP methods that the proxy should reject (example --reject-methods='POST,PUT,PATCH').
- `--reject-paths`

  Regular expression for paths that the proxy should reject. Paths specified here will be rejected even accepted by --accept-paths.
- `--unix-socket`, `-u`

  Unix socket on which to run the proxy.
- `--www`, `-w`

  Also serve static files from the given directory under the specified prefix.
- `--www-prefix`, `-P`

  Prefix to serve static files under, if static file directory is specified.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes

- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k replace

Usage:

```
d8 k replace -f FILENAME [options]
```

Replace a resource by file name or stdin

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--cascade`

  Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents (e.g. Pods created by a ReplicationController). Defaults to background.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  The files that contain the configurations to replace.
- `--force`

If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of some resources may result in inconsistency or data loss and requires confirmation.

- `--grace-period`

  Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set to 0 when --force is true (force deletion).
- `--kustomize`, `-k`

  Process a kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--raw`

  Raw URI to PUT to the server. Uses the transport specified by the kubeconfig file.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--subresource`

  If specified, replace will operate on the subresource of the requested object.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--timeout`

  The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.
- `--wait`

  If true, wait for resources to be gone before returning. This waits for finalizers.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout

Usage:

```
d8 k rollout SUBCOMMAND [options]
```

Manage the rollout of a resource

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`
  The address and port of the Kubernetes API server
- `--tls-server-name`
  Server name to use for server certificate validation. If it is not provided, the hostname used to

contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout history

Usage:

```
d8 k rollout SUBCOMMAND history (TYPE NAME | TYPE/NAME) [flags] [options]
```

View rollout history

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--revision`

  See the details, including podTemplate of the revision specified
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout pause

Usage:

```
d8 k rollout SUBCOMMAND pause RESOURCE [options]
```

Mark the provided resource as paused

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS

- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout restart

Usage:

```
d8 k rollout SUBCOMMAND restart RESOURCE [options]
```

Restart a resource

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.

- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout resume

Usage:

```
d8 k rollout SUBCOMMAND resume RESOURCE [options]
```

Resume a paused resource

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.

- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

Password for basic authentication to the API server

- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout status

Usage:

```
d8 k rollout SUBCOMMAND status (TYPE NAME | TYPE/NAME) [flags] [options]
```

Show the status of the rollout

Options

- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--revision`

  Pin to a specific revision for showing its status. Defaults to 0 (last revision).
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--timeout`

  The length of time to wait before ending watch, zero means never. Any other values should contain a corresponding time unit (e.g. 1s, 2m, 3h).
- `--watch`, `-w`

  Watch the status of the rollout until it's done.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use

- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k rollout undo

Usage:

```
d8 k rollout SUBCOMMAND undo (TYPE NAME | TYPE/NAME) [flags] [options]
```

Undo a previous rollout

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--to-revision`

  The revision to rollback to. Default to 0 (last revision).

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.

- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k run

Usage:

```
d8 k run NAME --image=image [--env="key=value"] [--port=port] [--dry-run=server|client] [--
overrides=inline-json] [--command] -- [COMMAND] [args...] [options]
```

Run a particular image on the cluster

Options

- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--annotations`

  Annotations to apply to the pod.
- `--attach`

  If true, wait for the Pod to start running, and then attach to the Pod as if 'kubectl attach ...' were called. Default false, unless '-i/--stdin' is set, in which case the default is true. With '--restart=Never' the exit code of the container process is returned.
- `--cascade`

  Must be "background", "orphan", or "foreground". Selects the deletion cascading strategy for the dependents (e.g. Pods created by a ReplicationController). Defaults to background.
- `--command`

  If true and extra arguments are present, use them as the 'command' field in the container, rather than the 'args' field which is the default.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--env`

  Environment variables to set in the container.
- `--expose`

  If true, create a ClusterIP service associated with the pod. Requires `--port`.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  to use to replace the resource.
- `--force`

  If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of some resources may result in inconsistency or data loss and requires confirmation.
- `--grace-period`

  Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set to 0 when --force is true (force deletion).
- `--image`

  The image for the container to run.
- `--image-pull-policy`

  The image pull policy for the container. If left empty, this value will not be specified by the client and defaulted by the server.
- `--kustomize`, `-k`

  Process a kustomization directory. This flag can't be used together with -f or -R.
- `--labels`, `-l`

  Comma separated labels to apply to the pod. Will override previous values.
- `--leave-stdin-open`

If the pod is started in interactive mode or with stdin, leave stdin open after the first attach completes. By default, stdin will be closed after the first attach completes.

- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--override-type`

  The method used to override the generated object: json, merge, or strategic.
- `--overrides`

  An inline JSON override for the generated object. If this is non-empty, it is used to override the generated object. Requires that the object supply a valid apiVersion field.
- `--pod-running-timeout`

  The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running
- `--port`

  The port that this container exposes.
- `--privileged`

  If true, run the container in privileged mode.
- `--quiet`, `-q`

  If true, suppress prompt messages.
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--restart`

  The restart policy for this Pod. Legal values [Always, OnFailure, Never].
- `--rm`

  If true, delete the pod after it exits. Only valid when attaching to the container, e.g. with '--attach' or with '-i/--stdin'.
- `--save-config`

  If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--stdin`, `-i`

  Keep stdin open on the container in the pod, even if nothing is attached.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--timeout`

  The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object
- `--tty`, `-t`

  Allocate a TTY for the container in the pod.
- `--wait`

  If true, wait for resources to be gone before returning. This waits for finalizers.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity

- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k scale

Usage:

```
d8 k scale [--resource-version=version] [--current-replicas=count] --replicas=COUNT (-f FILENAME | TYPE
NAME) [options]
```

Set a new size for a deployment, replica set, or replication controller

Options

- `--all`

  Select all resources in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--current-replicas`

  Precondition for current size. Requires that the current size of the resource match this value in order to scale. -1 (default) for no condition.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to set a new size
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--replicas`

  The new desired number of replicas. Required.
- `--resource-version`

  Precondition for resource version. Requires that the current resource version match this value in order to scale.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

- `--timeout`

  The length of time to wait before giving up on a scale operation, zero means don't wait. Any other values should contain a corresponding time unit (e.g. 1s, 2m, 3h).

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to

contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set

Usage:

```
d8 k set SUBCOMMAND [options]
```

Set specific features on objects

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes

- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set env

Usage:

```
d8 k set SUBCOMMAND env RESOURCE/NAME KEY_1=VAL_1 ... KEY_N=VAL_N [options]
```

Update environment variables on a pod template

Options

- `--all`

  If true, select all resources in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--containers`, `-c`

  The names of containers in the selected pod templates to change - may use wildcards
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--env`, `-e`

  Specify a key-value pair for an environment variable to set into each container.
- `--field-manager`

  Name of the manager used to track field ownership.

- `--filename`, `-f`

  Filename, directory, or URL to files the resource to update the env
- `--from`

  The name of a resource from which to inject environment variables
- `--keys`

  Comma-separated list of keys to import from specified resource
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--list`

  If true, display the environment and any changes in the standard format. this flag will removed when we have kubectl view env.
- `--local`

  If true, set env will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--overwrite`

  If true, allow environment to be overwritten, otherwise reject updates that overwrite existing environment.
- `--prefix`

  Prefix to append to variable names
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--resolve`

  If true, show secret or configmap references when listing variables
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set image

Usage:

```
d8 k set SUBCOMMAND image (-f FILENAME | TYPE NAME) CONTAINER_NAME_1=CONTAINER_IMAGE_1 ...
CONTAINER_NAME_N=CONTAINER_IMAGE_N [options]
```

Update the image of a pod template

Options

- `--all`

  Select all resources, in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--local`

  If true, set image will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS

- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set resources

Usage:

```
d8 k set SUBCOMMAND resources (-f FILENAME | TYPE NAME)  ([--limits=LIMITS & --requests=REQUESTS]
[options]
```

Update resource requests/limits on objects with pod templates

Options

- `--all`

  Select all resources, in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--containers`, `-c`

  The names of containers in the selected pod templates to change, all containers are selected by default - may use wildcards
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--limits`

  The resource requirement requests for this container. For example, 'cpu=100m,memory=256Mi'. Note that server side components may assign requests depending on the server configuration, such as limit ranges.
- `--local`

  If true, set resources will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--requests`

  The resource requirement requests for this container. For example, 'cpu=100m,memory=256Mi'. Note that server side components may assign requests depending on the server configuration, such as limit ranges.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.

- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set selector

Usage:

```
d8 k set SUBCOMMAND selector (-f FILENAME | TYPE NAME) EXPRESSIONS [--resource-version=version] [options]
```

Set the selector on a resource

Options

- `--all`

  Select all resources in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  identifying the resource.
- `--local`

  If true, annotation will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--resource-version`

  If non-empty, the selectors update will only succeed if this is the current resource-version for the object. Only valid when specifying a single resource.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set serviceaccount

Alias: `sa`

Usage:

```
d8 k set SUBCOMMAND serviceaccount (-f FILENAME | TYPE NAME) SERVICE_ACCOUNT [options]
```

Update the service account of a resource

Options

- `--all`

  Select all resources, in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files identifying the resource to get from a server.
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--local`

  If true, set serviceaccount will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--record`

  Record current kubectl command in the resource annotation. If set to false, do not record the command. If set to true, record the command. If not set, default to updating the existing annotation value only if one already exists.
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.

- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k set subject

Usage:

```
d8 k set SUBCOMMAND subject (-f FILENAME | TYPE NAME) [--user=username] [--group=groupname] [--
serviceaccount=namespace:serviceaccountname] [--dry-run=server|client|none] [options]
```

Update the user, group, or service account in a role binding or cluster role binding

Options

- `--all`

  Select all resources, in the namespace of the specified resource types
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--filename`, `-f`

  Filename, directory, or URL to files the resource to update the subjects
- `--group`

  Groups to bind to the role
- `--kustomize`, `-k`

  Process the kustomization directory. This flag can't be used together with -f or -R.
- `--local`

  If true, set subject will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--serviceaccount`

  Service accounts to bind to the role
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--user`

  Usernames to bind to the role

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

Group to impersonate for the operation, this flag can be repeated to specify multiple groups.

- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text

log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k taint

Usage:

```
d8 k taint NODE NAME KEY_1=VAL_1:TAINT_EFFECT_1 ... KEY_N=VAL_N:TAINT_EFFECT_N [options]
```

Update the taints on one or more nodes

Options

- `--all`

  Select all nodes in the cluster
- `--allow-missing-template-keys`

  If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.
- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.
- `--field-manager`

  Name of the manager used to track field ownership.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--overwrite`

  If true, allow taints to be overwritten, otherwise reject taint updates that overwrite existing taints.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--validate`

  Must be one of: strict (or true), warn, ignore (or false). "true" or "strict" will use a schema to validate the input and fail the request if invalid. It will perform server side validation if ServerSideFieldValidation is enabled on the api-server, but will fall back to less reliable client-side validation if not. "warn" will warn about unknown or duplicate fields without blocking the request if server-side field validation is enabled on the API server, and behave as "ignore" otherwise. "false" or "ignore" will not perform any schema validation, silently dropping any unknown or duplicate fields.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k top

Usage:

```
d8 k top [options]
```

Display resource (CPU/memory) usage

Global options

- `--as`
  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation.
- `--cache-dir`
  Default cache directory
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`
  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--match-server-version`
  Require server version to match client version
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--profile`
  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`
  Name of the file to write the profile to
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.

- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k top node

Aliases: `nodes`, `no`

Usage:

```
d8 k top node [NAME | -l label] [options]
```

Display resource (CPU/memory) usage of nodes

Options

- `--no-headers`

  If present, print output without headers
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-capacity`

  Print node resources based on Capacity instead of Allocatable(default) of the nodes.
- `--show-swap`

  Print node resources related to swap memory.
- `--sort-by`

  If non-empty, sort nodes list using specified field. The field can be either 'cpu' or 'memory'.
- `--use-protocol-buffers`

  Enables using protocol-buffers to access Metrics API.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k top pod

Aliases: `pods`, `po`

Usage:

```
d8 k top pod [NAME | -l label] [options]
```

Display resource (CPU/memory) usage of pods

Options

- `--all-namespaces`, `-A`

  If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.
- `--containers`

  If present, print usage of containers within a pod.
- `--field-selector`

  Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.
- `--no-headers`

  If present, print output without headers.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.
- `--show-swap`

  Print pod resources related to swap memory.
- `--sort-by`

  If non-empty, sort pods list using specified field. The field can be either 'cpu' or 'memory'.
- `--sum`

  Print the sum of the resource usage
- `--use-protocol-buffers`

  Enables using protocol-buffers to access Metrics API.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k uncordon

Usage:

```
d8 k uncordon NODE [options]
```

Mark node as schedulable

Options

- `--dry-run`

  Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', '!=', 'in', 'notin'.(e.g. -l key1=value1,key2=value2,key3 in (value3)). Matching objects must satisfy all of the specified label constraints.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k version

Usage:

```
d8 k version [options]
```

Print the client and server version information

Options

- `--client`

  If true, shows client version only (no server required).
- `--output`, `-o`

  One of 'yaml' or 'json'.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure

- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 k wait

Usage:

```
d8 k wait ([-f FILENAME] | resource.group/resource.name | resource.group [(-l label | --all)]) [--
for=create|--for=delete|--for condition=available|--for=jsonpath='{}'[=value]] [options]
```

Experimental: Wait for a specific condition on one or many resources

Options

- `--all`

  Select all resources in the namespace of the specified resource types
- `--all-namespaces`, `-A`

  If present, list the requested object(s) across all namespaces. Namespace in current context is ignored even if specified with --namespace.
- `--allow-missing-template-keys`

If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.

- `--field-selector`

  Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.
- `--filename`, `-f`

  identifying the resource.
- `--for`

  The condition to wait on: [create|delete|condition=condition-name[=condition-value]|jsonpath='{JSONPath expression}'=[JSONPath value]]. The default condition-value is true. Condition values are compared after Unicode simple case folding, which is a more general form of case-insensitivity.
- `--local`

  If true, annotation will NOT contact api-server but run locally.
- `--output`, `-o`

  Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).
- `--recursive`, `-R`

  Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same directory.
- `--selector`, `-l`

  Selector (label query) to filter on, supports '=', '==', and '!='.(e.g. -l key1=value1,key2=value2)
- `--show-managed-fields`

  If true, keep the managedFields when printing objects in JSON or YAML format.
- `--template`

  Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].
- `--timeout`

  The length of time to wait before giving up. Zero means check once and don't wait, negative means wait for a week.

Global options

- `--as`

  Username to impersonate for the operation. User could be a regular user or a service account in a namespace.
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation.
- `--cache-dir`

  Default cache directory
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server

- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--kuberc`

  Path to the kuberc file to use for preferences. This can be disabled by exporting KUBECTL_KUBERC=false feature gate or turning off the feature KUBERC=off.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--match-server-version`

  Require server version to match client version
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--profile`

  Name of profile to capture. One of (none|cpu|heap|goroutine|threadcreate|block|mutex)
- `--profile-output`

  Name of the file to write the profile to
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  Server name to use for server certificate validation. If it is not provided, the hostname used to contact the server is used
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)
- `--warnings-as-errors`

  Treat warnings received from the server as errors and exit with a non-zero exit code

d8 mirror

Copy Deckhouse Kubernetes Platform distribution to the local filesystem or third-party registry

d8 mirror pull

Usage:

```
d8 mirror pull <images-bundle-path> [options]
```

Copy Deckhouse Kubernetes Platform distribution to the local filesystem

Options

- `--deckhouse-tag`

Specific Deckhouse build tag to pull. Conflicts with --since-version and --include-platform. If registry contains release channel image for specified tag, all release channels in the bundle will be pointed to it.

- `--dry-run`

  Print what would be pulled without downloading any images. Useful for fast validation of flags and filters.
- `--exclude-module`, `-e`

  Blacklist specific modules from downloading. Format is "module-name[@version]". Use one flag per each module. Overridden by use of --include-module."
- `--exclude-package`

  Blacklist specific packages from downloading. Format is "package-name[@version]". Use one flag per each package. Overridden by use of --include-package.
- `--force`

  Overwrite existing bundle packages if they are conflicting with current pull operation.
- `--gost-digest`

  Calculate GOST R 34.11-2012 STREEBOG digest for downloaded bundle
- `--ignore-suspend`

  Ignore suspended release channels and continue mirroring. Use with caution.
- `--images-bundle-chunk-size`, `-c`

  Split resulting bundle file into chunks of at most N gigabytes
- `--include-module`, `-i`

  Whitelist specific modules for downloading. Use one flag per each module. Disables blacklisting by --exclude-module."

  Semver constraints (caret, tilde, range) keep only the highest patch in each (major, minor) series, mirroring how platform releases are discovered.
  Versions explicitly named with an inclusive boundary operator (>= or <=) are always preserved — that boundary is part of the user's request and must round-trip even when a newer patch exists in the same minor.
  Use the exact-tag form (=) when you need a specific older patch unconditionally.

  A bare version with no operator (module-name@1.3.0) means "this version or newer within the same major line" and keeps the latest patch per minor. It expands to >=X.Y.Z <(major+1).0.0, treating major 0 like any other major: module-name@0.4.0 spans the whole 0.x line (>=0.4.0 <1.0.0), NOT just the 0.4 minor as a caret (^0.4.0 = >=0.4.0 <0.5.0) would. Prefer this bare form for step-by-step upgrades — it captures every intermediate minor and needs no shell quoting.

  Shell note: >= and <= contain the redirection metacharacters > and <, so an unquoted module-name@>=1.3.0 is mangled by bash/zsh (it strips the operator and redirects into a file named "=1.3.0", leaving the module with an empty version). Quote the whole value when you use these operators: --include-module 'module-name@>=1.3.0'. The bare-version form above avoids this entirely.

  Example:
  Available versions for : v1.0.0, v1.1.0, v1.2.0, v1.3.0, v1.3.3, v1.4.0, v1.4.1

  module-name@1.3.0 → bare version, expands to >=1.3.0 <2.0.0 (same major line): keep latest patch per minor — includes v1.3.3 (1.3.x) and v1.4.1 (1.4.x). Versions currently pinned by release channels are pulled in addition.

  module-name@~1.3.0 → semver ~ constraint (>=1.3.0 <1.4.0): keep latest patch per minor in range — includes v1.3.3. Versions currently pinned by release channels are pulled in addition.

  module-name@^1.3.0 → semver ^ constraint (>=1.3.0 <2.0.0): keep latest patch per minor —

includes v1.3.3, v1.4.1. For a 0.x version the caret locks the minor (^0.4.0 = >=0.4.0 <0.5.0), so use the bare form instead when you want the whole 0.x line.

module-name@>=1.3.0 → range constraint with explicit >= anchor (quote in shell): keep latest patch per minor AND the named anchor v1.3.0 — includes v1.3.0 (anchor), v1.3.3 (1.3.x latest), v1.4.1 (1.4.x latest).

module-name@>=1.3.0 <=1.4.0 → both anchors honoured (quote in shell): includes v1.3.0, v1.3.3, v1.4.0; v1.4.1 is excluded by the upper bound.

module-name@=v1.3.0 → exact tag match: include only v1.3.0 and publish it to all release channels (alpha, beta, early-access, stable, rock-solid).

module-name@=bobV1 → exact tag match: include only bobV1 and publish it to all release channels (alpha, beta, early-access, stable, rock-solid).

module-name@=v1.3.0+stable → exact tag match: include only v1.3.0 and publish it to stable channel

0.x example — available versions for : v0.4.2, v0.4.4, v0.5.3, v0.6.1, v0.7.2
module-name@0.4.0 → bare version, expands to >=0.4.0 <1.0.0: keep latest patch per minor across the whole 0.x line — includes v0.4.4, v0.5.3, v0.6.1, v0.7.2.

- `--include-package`
  Whitelist specific packages for downloading. Use one flag per each package. Disables blacklisting by --exclude-package.

  Packages are mirrored exactly like modules (same name@version constraint dialect), but live under the packages/ registry segment with their release metadata under packages//version.
- `--include-platform`
  Select platform releases to download by a semver constraint expression, using the same dialect as --include-module's version part.
  Conflicts with --since-version and --deckhouse-tag.

  Semver constraints (caret, tilde, range) keep only the highest patch in each (major, minor) series, mirroring the release-discovery rules used for full pulls.
  Versions explicitly named with an inclusive boundary operator (>= or <=) are always preserved — that boundary is part of the user's request and must round-trip even when a newer patch exists in the same minor.
  Use the exact-tag form (=) when you need to pin a specific tag, optionally propagating it to a release channel via the +channel suffix.

  Examples (available platform versions: v1.63.x, v1.64.x, v1.65.x, v1.66.x, v1.67.x, v1.68.x, v1.69.x, v1.70.x, v1.71.x):

  --include-platform ">=1.64 <=1.68" → bounded range: latest patch per minor in v1.64..v1.68, anchors v1.64.0 and v1.68.0 always preserved if present in the registry.

  --include-platform "~1.65.0" → semver ~ constraint (>=1.65.0 <1.66.0): latest v1.65.x patch only.

  --include-platform "^1.65.0" → semver ^ constraint (>=1.65.0 <2.0.0): latest patch per minor starting at v1.65.x.

  --include-platform "1.65.0" → bare version, expands to >=1.65.0 <2.0.0 (same major line): keep

latest patch per minor starting at v1.65.x. Same result as ^1.65.0 here because platform is always major >= 1; shorthand kept for parity with --include-module.

--include-platform "=v1.65.3" → exact-tag pin: only v1.65.3 is pulled and propagated to all default release channels, just like --deckhouse-tag.

--include-platform "=v1.65.3+stable" → exact-tag pin with channel suffix: only v1.65.3 is pulled (channel propagation matches --deckhouse-tag).

- `--insecure`

  Interact with registries over HTTP.
- `--installer-tag`

  Specific Deckhouse installer build tag to pull. If not specified, the latest tag for the installer will be pulled from the registry path.
- `--license`, `-l`

  Deckhouse license key. Shortcut for --source-login=license-token --source-password=<>.
- `--modules-path-suffix`

  Suffix to append to source repo path to locate modules.
- `--no-installer`

  Do not pull Deckhouse installer into bundle.
- `--no-modules`

  Do not pull Deckhouse modules into bundle.
- `--no-packages`

  Do not pull Deckhouse packages into bundle.
- `--no-platform`

  Do not pull Deckhouse Kubernetes Platform into bundle.
- `--no-pull-resume`

  Do not continue last unfinished pull operation and start from scratch.
- `--no-security-db`

  Do not pull security databases into bundle.
- `--only-extra-images`

  Pull only extra images for modules (additional images like security databases, scanners, etc.) without pulling main module images.
- `--proxy-registry`

  Pull from a proxy/caching registry that does not implement the registry catalog API.

  Instead of calling the registry's "list tags" endpoint (which proxy registries typically return empty), this mode probes individual tags by incrementing patch -> minor -> major from the version explicitly named via --include-platform / --include-module. The probe stops once both a new patch and a new minor of the current major fail to resolve, then attempts the next major; if that also fails the probe terminates and downloads what was discovered.

  Requires --include-platform when platform is not skipped via --no-platform, and at least one --include-module when modules are not skipped via --no-modules. --exclude-module and --no-platform are respected.

  Cannot be combined with --deckhouse-tag or --since-version (use --include-platform's lower bound instead).
- `--since-version`

  Minimal Deckhouse release to pull. Ignored if above current Rock Solid release. Conflicts with --deckhouse-tag.
- `--skip-vex-images`

  Do not pull VEX images.
- `--source`

  Source registry to pull Deckhouse images from.

- `--source-login`

  Source registry login.
- `--source-password`

  Source registry password.
- `--tls-skip-verify`

  Disable TLS certificate validation.
- `--tmp-dir`

  Path to a temporary directory to use for image pulling and pushing. All processing is done in this directory, so make sure there is enough free disk space to accommodate the entire bundle you are downloading;
- `--verbose-summary`

  List every module and package in the end-of-pull summary with its resolved versions (and VEX count, when present), instead of just the totals. Output only - it does not change what is pulled.

d8 mirror push

Usage:

```
d8 mirror push [images-bundle-path] <registry> [options]
```

Copy Deckhouse Kubernetes Platform distribution to the third-party registry

Options

- `--file`

  `Path` to a tar or chunked package to push. May be repeated to push multiple packages. Can be used instead of the bundle directory argument or combined with it.
- `--insecure`

  Interact with registries over HTTP.
- `--modules-path-suffix`

  Suffix to append to source repo path to locate modules.
- `--registry-login`, `-u`

  Username to log into the target registry.
- `--registry-password`, `-p`

  Password to log into the target registry.
- `--tls-skip-verify`

  Disable TLS certificate validation.
- `--tmp-dir`

  Path to a temporary directory to use for image pulling and pushing. All processing is done in this directory, so make sure there is enough free disk space to accommodate the entire bundle you are downloading;

d8 network

Alias: `n`

A group of commands to operate network related tasks in The Deckhouse Ecosystem.

d8 network cni-migration

Usage:

```
d8 network cni-migration [options]
```

A group of commands to switch CNI in the cluster

d8 network cni-migration cleanup

Usage:

```
d8 network cni-migration cleanup [options]
```

Cleans up resources created during CNI switching

d8 network cni-migration switch

Usage:

```
d8 network cni-migration switch [options]
```

Initiates the CNI switching

Options

- `--to-cni`
  Target CNI provider to switch to. Supported values: cilium, flannel, simple-bridge

d8 network cni-migration watch

Usage:

```
d8 network cni-migration watch [options]
```

Monitors the CNI switching progress

d8 package

Package build and bootstrap tool for containerized packages

d8 package bootstrap

Usage:

```
d8 package bootstrap [options]
```

Bootstrap a new package (module or application)

d8 package bootstrap application

Usage:

```
d8 package bootstrap application <name> [options]
```

Bootstrap a new application package

Options

- `--extended`
  Generate extended files in the package
- `--hooks`
  Generate hooks in the package
- `--output`, `-o`
  Custom path for the package (default: /)
- `--werf`
  Use werf file for images

d8 package bootstrap module

Usage:

```
d8 package bootstrap module <name> [options]
```

Bootstrap a new module package

Options

- `--extended`

  Generate extended files in the package
- `--hooks`

  Generate hooks in the package
- `--output`, `-o`

  Custom path for the package (default: /)
- `--werf`

  Use werf file for images

d8 package build

Usage:

```
d8 package build --version <version> [--repository <repository>] [--force] [options]
```

Build and push a package to a container registry

Options

- `--debug`

  enable debug logging
- `--force`, `-f`

  force update version in registry
- `--repo`, `-r`

  Repository (env: PACKAGE_BUILD_REPOSITORY)
- `--sign`

  sign package with certificate and key. --sign-cert and --sign-key are required
- `--sign-cert`

  sign certificate path or base64 string (env: PACKAGE_BUILD_SIGN_CERT)
- `--sign-key`

  sign key path or base64 string or vault url (env: PACKAGE_BUILD_SIGN_KEY)
- `--token`, `-t`

  Registry token (env: PACKAGE_BUILD_REPOSITORY_TOKEN)
- `--user`, `-u`

  Registry user (env: PACKAGE_BUILD_REPOSITORY_USER)
- `--version`, `-v`

  Package version

d8 package render

Usage:

```
d8 package render [options]
```

Render package templates to Kubernetes manifests

Options

- `--file`

  Show only manifests rendered from the template with this file name
- `--render-file`

  Write the clean render (without '# Source:' headers) to this file path

d8 package verify

Usage:

```
d8 package verify [options]
```

Verify a package structure and configuration

Options

- `--hide-warnings`

  Hide warning-level messages
- `--lint-config`

  Path to lint config file
- `--remote`, `-r`

  Verify a package image from a registry reference
- `--show-ignored`

  Show ignored-level messages

d8 snapshot

Snapshot operations (create, delete, download, restore, upload, get)

d8 snapshot create

Usage:

```
d8 snapshot create <name> [options]
```

Create a Snapshot of a namespace

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--namespace`, `-n`

  namespace to create the Snapshot in (defaults to the kubeconfig context namespace)
- `--output`, `-o`

  Output format: name|json|yaml
- `--selector`, `-l`

  capture only objects matching this label selector (e.g. app=demo,tier=db); sets spec.resourceSelector.matchLabels
- `--timeout`

  timeout for --wait
- `--wait`

  wait until the Snapshot reports Ready

d8 snapshot delete

Usage:

```
d8 snapshot delete [name...] [options]
```

Delete one or more Snapshots

Options

- `--all`

  delete all Snapshots in the namespace
- `--context`

  The name of the kubeconfig context to use
- `--ignore-not-found`

  do not error if a named Snapshot does not exist
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--namespace`, `-n`

namespace of the Snapshots (defaults to the kubeconfig context namespace)

- `--selector`, `-l`

  delete Snapshots matching this label selector (e.g. app=demo,tier=db)
- `--timeout`

  timeout for --wait
- `--wait`

  wait until the Snapshots are fully deleted

d8 snapshot describe

Usage:

```
d8 snapshot describe [flags] <name> [options]
```

Show the snapshot hierarchy as a tree

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--namespace`, `-n`

  snapshot namespace (defaults to the kubeconfig context namespace)

d8 snapshot download

Usage:

```
d8 snapshot download [flags] <snapshot> [options]
```

Download a snapshot to a local directory tree

Options

- `--cleanup`

  delete the per-volume DataExport (and its server-side export chain) after each volume completes; --cleanup=false leaves them in the cluster for debugging
- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--max-parallel-downloads`

  global cap on concurrent whole-volume-stream downloads across all nodes (independent of --workers and --per-volume-concurrency)
- `--namespace`, `-n`

  snapshot namespace (required)
- `--node`

  restrict download to a single node subtree; format '/' (e.g. --node DemoVirtualDisk/bk-disk-a, --node Snapshot/my-snap); the generated snapshot CR name form (e.g. DemoVirtualDiskSnapshot/nss-child-abc) is still accepted
- `--output`, `-o`

  root output directory (required)
- `--per-volume-concurrency`

  maximum parallel chunk/file downloads per volume
- `--ttl`

  DataExport TTL (e.g. 2h, 30m)
- `--volume-compression`

  volume compression codec (none, zstd); block volumes: data.bin[.]; filesystem volumes: per-file

compressed entries inside an uncompressed data.tar container

- `--volume-compression-level`

  compression level for the selected codec (0 = codec default; ignored when --volume-compression=none)
- `--workers`

  maximum number of nodes downloaded concurrently

d8 snapshot get

Aliases: `list`, `ls`

Usage:

```
d8 snapshot get [options]
```

List Snapshot resources in the cluster

Options

- `--all-namespaces`, `-A`

  list Snapshots across all namespaces
- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--namespace`, `-n`

  namespace to list Snapshots from (defaults to the kubeconfig context namespace)
- `--output`, `-o`

  Output format: table|json|yaml

d8 snapshot local

Usage:

```
d8 snapshot local [options]
```

Operate on a locally downloaded snapshot archive (no kubeconfig required)

d8 snapshot local describe

Usage:

```
d8 snapshot local describe <DIR> [options]
```

Show the structure of a locally downloaded snapshot archive as a tree

Options

- `--allow-unauthenticated-legacy`

  allow trusted pre-version archives with unauthenticated snapshot.yaml metadata (unsafe compatibility mode)

d8 snapshot local get

Usage:

```
d8 snapshot local get <DIR> [options]
```

Print a one-line summary of a locally downloaded snapshot archive

Options

- `--allow-unauthenticated-legacy`

  allow trusted pre-version archives with unauthenticated snapshot.yaml metadata (unsafe

compatibility mode)

d8 snapshot restore

Usage:

```
d8 snapshot restore [flags] <snapshot> [options]
```

Restore a snapshot in-place into its namespace

Options

- `--context`
  The name of the kubeconfig context to use
- `--dry-run`
  validate objects via DryRunAll without persisting; skips --wait (use to preflight a restore)
- `--edit`
  open resolved manifests in $KUBE_EDITOR/$EDITOR (falling back to vi) before applying; aborts on non-zero exit, unchanged, or empty content
- `--kubeconfig`, `-k`
  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--namespace`, `-n`
  snapshot namespace; also the restore target namespace (required)
- `--no-auto-edit`
  disable the one automatic editor session for interactive Kubernetes Invalid (HTTP 422) preflight responses
- `--node`
  restrict restore to a single node subtree; format '/' (e.g. --node DemoVirtualDisk/bk-disk-a); the generated snapshot CR name form (e.g. DemoVirtualDiskSnapshot/nss-child-abc123) is still accepted
- `--node-api-version`
  disambiguate --node by exact API version: 'v1' for core resources or '/' for named groups (requires --node)
- `--object`
  restrict a --scope node restore to a single captured object; format '/' (requires --scope node)
- `--scope`
  restore scope: 'subtree' (default) compiles the addressed node and its whole subtree; 'node' compiles only the addressed node itself
- `--timeout`
  timeout for the --wait Bound check
- `--wait`
  wait for manifest PVCs to become Bound; a dormant WaitForFirstConsumer PVC with no selected node, live consumer, or provisioning signal completes without polling

d8 snapshot upload

Alias: `import`

Usage:

```
d8 snapshot upload [flags] [options]
```

Upload a downloaded snapshot archive into a namespace

Options

- `--allow-existing`
  downgrade namespace preflight conflict check to a warning (import-mode markers from a prior run are never conflicts regardless of this flag)
- `--allow-unauthenticated-legacy`

allow trusted pre-version archives whose snapshot.yaml metadata is unauthenticated (unsafe; explicit compatibility mode)

- `--context`

  The name of the kubeconfig context to use
- `--input`, `-i`

  root archive directory produced by 'd8 snapshot download' (required)
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--namespace`, `-n`

  target namespace to import into (required)
- `--node`

  restrict import to a single node subtree; format '/' (e.g. --node VolumeSnapshot/pvc-1)
- `--skip-unsupported-fs-entries`

  skip unsupported filesystem entries and report them after upload (causes data loss for skipped paths)
- `--timeout`

  timeout for per-node readiness/completion waits
- `--ttl`

  idle TTL for each data-leaf DataImport (e.g. 2h, 30m); must exceed the importer's provisioning and post-upload completion time
- `--workers`

  maximum number of data-leaf volume uploads to run in parallel (fixed cap via errgroup.SetLimit; default 5)

d8 status

Get cluster status information

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 stronghold

Deckhouse Stronghold commands

d8 stronghold audit

Usage:

```
d8 stronghold [command] [args...] audit [options]
```

Interact with audit devices

d8 stronghold audit disable

Usage:

```
d8 stronghold [command] [args...] audit disable [options]
```

Disables an audit device

d8 stronghold audit enable

Usage:

```
d8 stronghold [command] [args...] audit enable [options]
```

Enables an audit device

d8 stronghold audit list

Usage:

```
d8 stronghold [command] [args...] audit list [options]
```

Lists enabled audit devices

d8 stronghold auth

Usage:

```
d8 stronghold [command] [args...] auth [options]
```

Interact with auth methods

d8 stronghold auth disable

Usage:

```
d8 stronghold [command] [args...] auth disable [options]
```

Disables an auth method

d8 stronghold auth enable

Usage:

```
d8 stronghold [command] [args...] auth enable [options]
```

Enables a new auth method

d8 stronghold auth help

Usage:

```
d8 stronghold [command] [args...] auth help [options]
```

Prints usage for an auth method

d8 stronghold auth list

Usage:

```
d8 stronghold [command] [args...] auth list [options]
```

Lists enabled auth methods

d8 stronghold auth move

Usage:

```
d8 stronghold [command] [args...] auth move [options]
```

Move an auth method to a new path

d8 stronghold auth tune

Usage:

```
d8 stronghold [command] [args...] auth tune [options]
```

Tunes an auth method configuration

d8 stronghold debug

Usage:

```
d8 stronghold [command] [args...] debug [options]
```

Runs the debug command

d8 stronghold delete

Usage:

```
d8 stronghold [command] [args...] delete [options]
```

Delete secrets and configuration

d8 stronghold events

Usage:

```
d8 stronghold [command] [args...] events [options]
```

d8 stronghold events subscribe

Usage:

```
d8 stronghold [command] [args...] events subscribe [options]
```

Subscribe to events

d8 stronghold kv

Usage:

```
d8 stronghold [command] [args...] kv [options]
```

Interact with Stronghold's Key-Value storage

d8 stronghold kv delete

Usage:

```
d8 stronghold [command] [args...] kv delete [options]
```

Deletes versions in the KV store

d8 stronghold kv destroy

Usage:

```
d8 stronghold [command] [args...] kv destroy [options]
```

Permanently removes one or more versions in the KV store

d8 stronghold kv enable-versioning

Usage:

```
d8 stronghold [command] [args...] kv enable-versioning [options]
```

Turns on versioning for a KV store

d8 stronghold kv get

Usage:

```
d8 stronghold [command] [args...] kv get [options]
```

Retrieves data from the KV store

d8 stronghold kv list

Usage:

```
d8 stronghold [command] [args...] kv list [options]
```

List data or secrets

d8 stronghold kv metadata

Usage:

```
d8 stronghold [command] [args...] kv metadata [options]
```

Interact with Stronghold's Key-Value storage

d8 stronghold kv metadata delete

Usage:

```
d8 stronghold [command] [args...] kv metadata delete [options]
```

Deletes all versions and metadata for a key in the KV store

d8 stronghold kv metadata get

Usage:

```
d8 stronghold [command] [args...] kv metadata get [options]
```

Retrieves key metadata from the KV store

d8 stronghold kv metadata patch

Usage:

```
d8 stronghold [command] [args...] kv metadata patch [options]
```

Patches key settings in the KV store

d8 stronghold kv metadata put

Usage:

```
d8 stronghold [command] [args...] kv metadata put [options]
```

Sets or updates key settings in the KV store

d8 stronghold kv patch

Usage:

```
d8 stronghold [command] [args...] kv patch [options]
```

Sets or updates data in the KV store without overwriting

d8 stronghold kv put

Usage:

```
d8 stronghold [command] [args...] kv put [options]
```

Sets or updates data in the KV store

d8 stronghold kv rollback

Usage:

```
d8 stronghold [command] [args...] kv rollback [options]
```

Rolls back to a previous version of data

d8 stronghold kv undelete

Usage:

```
d8 stronghold [command] [args...] kv undelete [options]
```

Undeletes versions in the KV store

d8 stronghold lease

Usage:

```
d8 stronghold [command] [args...] lease [options]
```

Interact with leases

d8 stronghold lease lookup

Usage:

```
d8 stronghold [command] [args...] lease lookup [options]
```

Lookup the lease of a secret

d8 stronghold lease renew

Usage:

```
d8 stronghold [command] [args...] lease renew [options]
```

Renews the lease of a secret

d8 stronghold lease revoke

Usage:

```
d8 stronghold [command] [args...] lease revoke [options]
```

Revokes leases and secrets

d8 stronghold list

Usage:

```
d8 stronghold [command] [args...] list [options]
```

List data or secrets

d8 stronghold login

Usage:

```
d8 stronghold [command] [args...] login [options]
```

Authenticate locally

d8 stronghold monitor

Usage:

```
d8 stronghold [command] [args...] monitor [options]
```

Stream log messages from a Stronghold server

d8 stronghold namespace

Usage:

```
d8 stronghold [command] [args...] namespace [options]
```

Interact with namespaces

d8 stronghold namespace create

Usage:

```
d8 stronghold [command] [args...] namespace create [options]
```

Create a new namespace

d8 stronghold namespace delete

Usage:

```
d8 stronghold [command] [args...] namespace delete [options]
```

Delete an existing namespace

d8 stronghold namespace list

Usage:

```
d8 stronghold [command] [args...] namespace list [options]
```

List child namespaces

d8 stronghold namespace lock

Usage:

```
d8 stronghold [command] [args...] namespace lock [options]
```

Lock the API for particular namespaces

d8 stronghold namespace lookup

Usage:

```
d8 stronghold [command] [args...] namespace lookup [options]
```

Look up an existing namespace

d8 stronghold namespace patch

Usage:

```
d8 stronghold [command] [args...] namespace patch [options]
```

Patch an existing namespace

d8 stronghold namespace unlock

Usage:

```
d8 stronghold [command] [args...] namespace unlock [options]
```

Unlock the API for particular namespaces

d8 stronghold operator

Usage:

```
d8 stronghold [command] [args...] operator [options]
```

Perform operator-specific tasks

d8 stronghold operator generate-root

Usage:

```
d8 stronghold [command] [args...] operator generate-root [options]
```

Generates a new root, DR operation, or recovery token

d8 stronghold operator init

Usage:

```
d8 stronghold [command] [args...] operator init [options]
```

Initializes a server

d8 stronghold operator key-status

Usage:

```
d8 stronghold [command] [args...] operator key-status [options]
```

Provides information about the active encryption key

d8 stronghold operator members

Usage:

```
d8 stronghold [command] [args...] operator members [options]
```

Returns the list of nodes in the cluster

d8 stronghold operator raft

Usage:

```
d8 stronghold [command] [args...] operator raft [options]
```

Interact with Vault's raft storage backend

d8 stronghold operator raft autopilot

Usage:

```
d8 stronghold [command] [args...] operator raft autopilot [options]
```

d8 stronghold operator raft autopilot get-config

Usage:

```
d8 stronghold [command] [args...] operator raft autopilot get-config [options]
```

Returns the configuration of the autopilot subsystem under integrated storage

d8 stronghold operator raft autopilot set-config

Usage:

```
d8 stronghold [command] [args...] operator raft autopilot set-config [options]
```

Modify the configuration of the autopilot subsystem under integrated storage

d8 stronghold operator raft autopilot state

Usage:

```
d8 stronghold [command] [args...] operator raft autopilot state [options]
```

Displays the state of the raft cluster under integrated storage as seen by autopilot

d8 stronghold operator raft demote

Usage:

```
d8 stronghold [command] [args...] operator raft demote [options]
```

Demotes a voter to a permanent non-voter

d8 stronghold operator raft join

Usage:

```
d8 stronghold [command] [args...] operator raft join [options]
```

Joins a node to the Raft cluster

d8 stronghold operator raft list-peers

Usage:

```
d8 stronghold [command] [args...] operator raft list-peers [options]
```

Returns the Raft peer set

d8 stronghold operator raft promote

Usage:

```
d8 stronghold [command] [args...] operator raft promote [options]
```

Promotes a permanent non-voter to a voter

d8 stronghold operator raft remove-peer

Usage:

```
d8 stronghold [command] [args...] operator raft remove-peer [options]
```

Removes a node from the Raft cluster

d8 stronghold operator raft snapshot

Usage:

```
d8 stronghold [command] [args...] operator raft snapshot [options]
```

Restores and saves snapshots from the Raft cluster

d8 stronghold operator raft snapshot restore

Usage:

```
d8 stronghold [command] [args...] operator raft snapshot restore [options]
```

Installs the provided snapshot, returning the cluster to the state defined in it

d8 stronghold operator raft snapshot save

Usage:

```
d8 stronghold [command] [args...] operator raft snapshot save [options]
```

Saves a snapshot of the current state of the Raft cluster into a file

d8 stronghold operator rekey

Usage:

```
d8 stronghold [command] [args...] operator rekey [options]
```

Generates new unseal keys

d8 stronghold operator rotate

Usage:

```
d8 stronghold [command] [args...] operator rotate [options]
```

Rotates the underlying encryption key

d8 stronghold operator seal

Usage:

```
d8 stronghold [command] [args...] operator seal [options]
```

Seals the Stronghold server

d8 stronghold operator step-down

Usage:

```
d8 stronghold [command] [args...] operator step-down [options]
```

Forces Stronghold to resign active duty

d8 stronghold operator unseal

Usage:

```
d8 stronghold [command] [args...] operator unseal [options]
```

Unseals the Stronghold server

d8 stronghold operator usage

Usage:

```
d8 stronghold [command] [args...] operator usage [options]
```

Lists historical client counts

d8 stronghold patch

Usage:

```
d8 stronghold [command] [args...] patch [options]
```

Patch data, configuration, and secrets

d8 stronghold path-help

Usage:

```
d8 stronghold [command] [args...] path-help [options]
```

Retrieve API help for paths

d8 stronghold pki

Usage:

```
d8 stronghold [command] [args...] pki [options]
```

Interact with Stronghold's PKI Secrets Engine

d8 stronghold pki health-check

Usage:

```
d8 stronghold [command] [args...] pki health-check [options]
```

Check a PKI Secrets Engine mount's health and operational status

d8 stronghold pki issue

Usage:

```
d8 stronghold [command] [args...] pki issue [options]
```

Given a parent certificate, and a list of generation parameters, creates an issuer on a specified mount

d8 stronghold pki list-intermediates

Usage:

```
d8 stronghold [command] [args...] pki list-intermediates [options]
```

Determine which of a list of certificates, were issued by a given parent certificate

d8 stronghold pki reissue

Usage:

```
d8 stronghold [command] [args...] pki reissue [options]
```

Uses a parent certificate and a template certificate to create a new issuer on a child mount

d8 stronghold pki verify-sign

Usage:

```
d8 stronghold [command] [args...] pki verify-sign [options]
```

Check whether one certificate validates another specified certificate

d8 stronghold plugin

Usage:

```
d8 stronghold [command] [args...] plugin [options]
```

Interact with Stronghold plugins and catalog

d8 stronghold plugin deregister

Usage:

```
d8 stronghold [command] [args...] plugin deregister [options]
```

Deregister an existing plugin in the catalog

d8 stronghold plugin info

Usage:

```
d8 stronghold [command] [args...] plugin info [options]
```

Read information about a plugin in the catalog

d8 stronghold plugin list

Usage:

```
d8 stronghold [command] [args...] plugin list [options]
```

Lists available plugins

d8 stronghold plugin register

Usage:

```
d8 stronghold [command] [args...] plugin register [options]
```

Registers a new plugin in the catalog

d8 stronghold plugin reload

Usage:

```
d8 stronghold [command] [args...] plugin reload [options]
```

Reload mounted plugin backend

d8 stronghold plugin reload-status

Usage:

```
d8 stronghold [command] [args...] plugin reload-status [options]
```

Get the status of an active or recently completed global plugin reload

d8 stronghold policy

Usage:

```
d8 stronghold [command] [args...] policy [options]
```

Interact with policies

d8 stronghold policy delete

Usage:

```
d8 stronghold [command] [args...] policy delete [options]
```

Deletes a policy by name

d8 stronghold policy list

Usage:

```
d8 stronghold [command] [args...] policy list [options]
```

Lists the installed policies

d8 stronghold policy read

Usage:

```
d8 stronghold [command] [args...] policy read [options]
```

Prints the contents of a policy

d8 stronghold policy write

Usage:

```
d8 stronghold [command] [args...] policy write [options]
```

Uploads a named policy from a file

d8 stronghold print

Usage:

```
d8 stronghold [command] [args...] print [options]
```

Prints runtime configurations

d8 stronghold print token

Usage:

```
d8 stronghold [command] [args...] print token [options]
```

Prints token currently in use

d8 stronghold read

Usage:

```
d8 stronghold [command] [args...] read [options]
```

Read data and retrieves secrets

d8 stronghold secrets

Usage:

```
d8 stronghold [command] [args...] secrets [options]
```

Interact with secrets engines

d8 stronghold secrets disable

Usage:

```
d8 stronghold [command] [args...] secrets disable [options]
```

Disable a secret engine

d8 stronghold secrets enable

Usage:

```
d8 stronghold [command] [args...] secrets enable [options]
```

Enable a secrets engine

d8 stronghold secrets list

Usage:

```
d8 stronghold [command] [args...] secrets list [options]
```

List enabled secrets engines

d8 stronghold secrets move

Usage:

```
d8 stronghold [command] [args...] secrets move [options]
```

Move a secrets engine to a new path

d8 stronghold secrets tune

Usage:

```
d8 stronghold [command] [args...] secrets tune [options]
```

Tune a secrets engine configuration

d8 stronghold ssh

Usage:

```
d8 stronghold [command] [args...] ssh [options]
```

Initiate an SSH session

d8 stronghold status

Usage:

```
d8 stronghold [command] [args...] status [options]
```

Print seal and HA status

d8 stronghold token

Usage:

```
d8 stronghold [command] [args...] token [options]
```

Interact with tokens

d8 stronghold token capabilities

Usage:

```
d8 stronghold [command] [args...] token capabilities [options]
```

Print capabilities of a token on a path

d8 stronghold token create

Usage:

```
d8 stronghold [command] [args...] token create [options]
```

Create a new token

d8 stronghold token lookup

Usage:

```
d8 stronghold [command] [args...] token lookup [options]
```

Display information about a token

d8 stronghold token renew

Usage:

```
d8 stronghold [command] [args...] token renew [options]
```

Renew a token lease

d8 stronghold token revoke

Usage:

```
d8 stronghold [command] [args...] token revoke [options]
```

Revoke a token and its children

d8 stronghold transit

Usage:

```
d8 stronghold [command] [args...] transit [options]
```

Interact with Stronghold's Transit Secrets Engine

d8 stronghold transit import

Usage:

```
d8 stronghold [command] [args...] transit import [options]
```

Import a key into the Transit secrets engines.

d8 stronghold transit import-version

Usage:

```
d8 stronghold [command] [args...] transit import-version [options]
```

Import key material into a new key version in the Transit secrets engines.

d8 stronghold unwrap

Usage:

```
d8 stronghold [command] [args...] unwrap [options]
```

Unwrap a wrapped secret

d8 stronghold version

Usage:

```
d8 stronghold [command] [args...] version [options]
```

Prints the Stronghold CLI version

d8 stronghold version-history

Usage:

```
d8 stronghold [command] [args...] version-history [options]
```

Prints the version history of the target Stronghold server

d8 stronghold write

Usage:

```
d8 stronghold [command] [args...] write [options]
```

Write data, configuration, and secrets

d8 system

Aliases: `s`, `p`, `platform`

Operate system options. Aliases: s, p, platform

Options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system collect-debug-info

Usage:

```
d8 system collect-debug-info [flags] > deckhouse-debug-$(date +"%Y_%m_%d").tar.gz [options]
```

Collect debug info.

Options

- `--command-timeout`

  Timeout for each individual debug command execution
- `--exclude`

  Exclude specific files from the debug archive. Use comma-separated values
- `--list-exclude`, `-l`

  List all files that can be excluded from the debug archive
- `--request-interval`

  Minimum interval between debug command executions to avoid overloading the cluster (e.g. 200ms, 500ms, 1s). Zero disables rate limiting (default 0s)

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system edit

Usage:

```
d8 system edit [options]
```

Edit configuration files

Options

- `--editor`, `-e`

  Your favourite editor.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system edit cluster-configuration

Usage:

```
d8 system edit cluster-configuration [options]
```

Edit cluster-configuration.

Options

- `--editor`, `-e`

  Your favourite editor.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system edit provider-cluster-configuration

Usage:

```
d8 system edit provider-cluster-configuration [options]
```

Edit provider-cluster-configuration.

Options

- `--editor`, `-e`

  Your favourite editor.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system edit static-cluster-configuration

Usage:

```
d8 system edit static-cluster-configuration [options]
```

Edit static-cluster-configuration.

Options

- `--editor`, `-e`

  Your favourite editor.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system get

Usage:

```
d8 system get [options]
```

Get configuration files

Global options

- `--context`

  The name of the kubeconfig context to use

- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system get cluster-configuration

Usage:

```
d8 system get cluster-configuration [options]
```

Get cluster-configuration.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system get provider-cluster-configuration

Usage:

```
d8 system get provider-cluster-configuration [options]
```

Get provider-cluster-configuration.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system get static-cluster-configuration

Usage:

```
d8 system get static-cluster-configuration [options]
```

Get static-cluster-configuration.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system logs

Usage:

```
d8 system logs [options]
```

Inspect deckhouse-controller logs.

Options

- `--follow`, `-f`

  Specify if the logs should be streamed.
- `--since`

  Show logs newer than a relative duration like 5s, 2m, or 1h. Only one of since-time / since may be used.
- `--since-time`

Show logs after a specific timestamp, e.g. --since-time='2025-05-19 12:00:00'. Only one of since-time / since may be used.

- `--tail`

  Limit the number of lines in the output to the last N lines. Defaults to -1 with no limit.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system module

Usage:

```
d8 system module [options]
```

Operate the Deckhouse Kubernetes Platform modules

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system module apply-now

Usage:

```
d8 system module apply-now <module-name> <version> [options]
```

Apply a module release immediately, bypassing update windows.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system module approve

Usage:

```
d8 system module approve <module-name> <version> [options]
```

Approve a pending module release for Manual update policy.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system module disable

Usage:

```
d8 system module disable [options]
```

Disable module.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system module enable

Usage:

```
d8 system module enable [options]
```

Enable module.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system module list

Usage:

```
d8 system module list [options]
```

List enabled modules.

Options

- `--editor`, `-e`

  Your favourite editor.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system module maintenance

Usage:

```
d8 system module maintenance [options]
```

Enable or disable module maintenance mode.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system module maintenance disable

Usage:

```
d8 system module maintenance disable <module-name> [options]
```

Disable maintenance mode for a module using ModuleConfig resource.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system module maintenance enable

Usage:

```
d8 system module maintenance enable <module-name> [options]
```

Enable maintenance mode for a module using ModuleConfig resource.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system module snapshots

Usage:

```
d8 system module snapshots [options]
```

Dump shapshots.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system module values

Usage:

```
d8 system module values [options]
```

Dump values.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system package

Usage:

```
d8 system package [options]
```

Operate the Deckhouse Kubernetes Platform packages

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system package scan

Usage:

```
d8 system package scan <repository-name> [options]
```

Create a scan task for a PackageRepository

Options

- `--dry-run`

  Preview the operation without creating it
- `--name`

  Custom name for the PackageRepositoryOperation (auto-generated if not specified)
- `--timeout`

  Timeout for the scan operation

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system queue

Usage:

```
d8 system queue [options]
```

Dump queues.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system queue list

Usage:

```
d8 system queue list [options]
```

Dump all queues.

Options

- `--output`, `-o`

  Output format: json|yaml|text.
- `--show-empty`, `-e`

  Show empty queues.
- `--watch`, `-w`

  Watch for changes to the queue list.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 system queue main

Usage:

```
d8 system queue main [options]
```

Dump main queue.

Options

- `--output`, `-o`

  Output format: json|yaml|text.

Global options

- `--context`

  The name of the kubeconfig context to use
- `--kubeconfig`, `-k`

  KubeConfig of the cluster. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)

d8 tools

Alias: `t`

Various useful tools for operating in The Deckhouse Ecosystem.

d8 tools far-convert

Usage:

```
d8 tools far-convert [options]
```

Converts files with Falco rules to FalcoAuditRules CRD format

d8 tools gostsum

Usage:

```
d8 tools gostsum [options]
```

Calculate Streebog checksum of the data according to the RFC 6986 (GOST R 34.11-2012) specifications

Options

- `--hash-length`, `-l`

  Bit-length variant of the algorithm to use. Only 256 (default) and 512 bit lengths are supported.

d8 tools imagedigest

Usage:

```
d8 tools imagedigest [options]
```

Options

- `--debug`, `-d`

  Enable debug logging
- `--insecure`, `-i`

  Allow insecure connections to registries (skip TLS verification)
- `--json`

  Use JSON formatter for output logs

d8 tools imagedigest add

Usage:

```
d8 tools imagedigest add <image> [options]
```

Calculating and adding the image digest to the image metadata according to the GOST standard Streebog (GOST R 34.11-2012)

Global options

- `--debug`, `-d`

  Enable debug logging
- `--insecure`, `-i`

  Allow insecure connections to registries (skip TLS verification)
- `--json`

  Use JSON formatter for output logs

d8 tools imagedigest calculate

Usage:

```
d8 tools imagedigest calculate <image> [options]
```

Calculating the image digest according to the GOST standard Streebog (GOST R 34.11-2012)

Global options

- `--debug`, `-d`

  Enable debug logging
- `--insecure`, `-i`

  Allow insecure connections to registries (skip TLS verification)
- `--json`

  Use JSON formatter for output logs

d8 tools imagedigest calculate-from-file

Usage:

```
d8 tools imagedigest calculate-from-file <file> [options]
```

Calculating the file digest according to the GOST standard Streebog (GOST R 34.11-2012). For stdin use '-'

Global options

- `--debug`, `-d`

  Enable debug logging
- `--insecure`, `-i`

  Allow insecure connections to registries (skip TLS verification)
- `--json`

  Use JSON formatter for output logs

d8 tools imagedigest validate

Usage:

```
d8 tools imagedigest validate <image> [options]
```

Validating the image digest in the image metadata calculated according to the GOST standard Streebog (GOST R 34.11-2012)

Options

- `--fix`

  Fix GOST Image Digest if it is incorrect

Global options

- `--debug`, `-d`

  Enable debug logging
- `--insecure`, `-i`

  Allow insecure connections to registries (skip TLS verification)
- `--json`

  Use JSON formatter for output logs

d8 tools pki

Usage:

```
d8 tools pki [options]
```

Tools for working with control-plane PKI

d8 tools pki certs

Usage:

```
d8 tools pki certs [options]
```

Manage and inspect control-plane TLS certificates

d8 tools pki certs check

Usage:

```
d8 tools pki certs check [PATH] [options]
```

Check expiration of control-plane certificates

Options

- `--kubeconfig-dir`

  Directory containing kubeconfig files (full-scan mode only; defaults to the parent of --path)
- `--path`

  Directory containing the PKI certificates (used in full-scan mode only)

d8 tools pki certs renew

Usage:

```
d8 tools pki certs renew (all | PATH) [options]
```

Renew control-plane certificates and kubeconfig client certificates

Options

- `--dry-run`

  Run all without writing any files
- `--kubeconfig-dir`

  Directory containing kubeconfig files (defaults to the parent of --path)
- `--path`

  Directory containing the PKI certificates and CA files

d8 tools pki certs renew all

Usage:

```
d8 tools pki certs renew (all | PATH) all [options]
```

Renew all control-plane certificates and kubeconfig client certificates

Options

- `--dry-run`

  Run all without writing any files
- `--kubeconfig-dir`

  Directory containing kubeconfig files (defaults to the parent of --path)
- `--path`

  Directory containing the PKI certificates and CA files
- `--san`

  New IP SAN to add to serving certificates (e.g. the new master IP)

d8 tools sig-migrate

Usage:

```
d8 tools sig-migrate [options]
```

Migrate Kubernetes resources by adding and removing migration annotations

Options

- `--as`

  Specify a Kubernetes service account for the kubectl operations (impersonation).
- `--context`

  The name of the kubeconfig context to use.
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests. (default is $KUBECONFIG when it is set, $HOME/.kube/config otherwise)
- `--log-level`

  Set the log level (INFO, DEBUG, TRACE). Defaults to DEBUG.
- `--measure-stages`

  Print execution time for major migration stages.
- `--object`

  Process objects by identifier in format //. Use 'clusterwide' namespace for cluster-scoped resources. Resource name must match kubectl api-resources output.
- `--retry`

  Retry annotation for previously failed objects.
- `--threads`

  Number of worker threads for resource discovery and migration. Values <=0 use default.

d8 v

Alias: `virtualization`

Commands to work with Deckhouse Virtualization Platform.

Options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v ansible-inventory

Usage:

```
d8 v [command] [args...] ansible-inventory [options]
```

Generate ansible inventory from virtual machines (ansible inventory script compatible).

Options

- `--host`

  Return variables for specific host
- `--list`

  Return all hosts (default behavior if no arguments)
- `--namespace`, `-n`

  Namespace to list virtual machines from (overrides kubeconfig context namespace)
- `--output`, `-o`

  Output format: json, ini, or yaml (default: json)

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v collect-debug-info

Usage:

```
d8 v [command] [args...] collect-debug-info (VirtualMachine) [options]
```

Collect debug information for VM: configuration, events, and logs. Output is written as compressed archive to stdout.

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity

- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v console

Usage:

```
d8 v [command] [args...] console (VirtualMachine) [options]
```

Connect to a console of a virtual machine.

Options

- `--timeout`

  Duration to wait until console is successfully connected (e.g., 1s, 5m, 300s).

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname

used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v evict

Usage:

```
d8 v [command] [args...] evict (VirtualMachine) [options]
```

Evict a virtual machine.

Options

- `--create-only`, `-c`

  Set this flag to only create the action without status warnings or notifications.
- `--force`, `-f`

  Set this flag to force the operation.
- `--timeout`, `-t`

  Set this flag to change the timeout.
- `--wait`, `-w`

  Set this flag to wait for the operation to complete.

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.

- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v migrate

Usage:

```
d8 v [command] [args...] migrate (VirtualMachine) [options]
```

Migrate a virtual machine.

Options

- `--create-only`, `-c`

  Set this flag to only create the action without status warnings or notifications.
- `--force`, `-f`

  Set this flag to force the operation.
- `--target-node-name`

  Set the target node name for virtual machine migration.
- `--timeout`, `-t`

  Set this flag to change the timeout.
- `--wait`, `-w`

  Set this flag to wait for the operation to complete.

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation

- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v port-forward

Usage:

```
d8 v [command] [args...] port-forward name[.namespace] [protocol/]localPort[:targetPort]... [options]
```

Forward local ports to a virtual machine

Options

- `--address`

--address=: Set this to the address the local ports should be opened on

- `--stdio`

  --stdio=true: Set this to true to forward the tunnel to stdout/stdin; Only works with a single port

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v restart

Usage:

```
d8 v [command] [args...] restart (VirtualMachine) [options]
```

Restart a virtual machine.

Options

- `--create-only`, `-c`
  Set this flag to only create the action without status warnings or notifications.
- `--force`, `-f`
  Set this flag to force the operation.
- `--timeout`, `-t`
  Set this flag to change the timeout.
- `--wait`, `-w`
  Set this flag to wait for the operation to complete.

Global options

- `--as`
  Username to impersonate for the operation
- `--as-group`
  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`
  UID to impersonate for the operation
- `--certificate-authority`
  Path to a cert file for the certificate authority
- `--client-certificate`
  Path to a client certificate file for TLS
- `--client-key`
  Path to a client key file for TLS
- `--cluster`
  The name of the kubeconfig cluster to use
- `--context`
  The name of the kubeconfig context to use
- `--disable-compression`
  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`
  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`
  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`
  Maximum number of seconds between log flushes
- `--namespace`, `-n`
  If present, the namespace scope for this CLI request
- `--password`
  Password for basic authentication to the API server
- `--proxy-url`
  If provided, this URL will be used to connect via proxy
- `--request-timeout`
  The length of time to wait before giving up on a single server request. Non-zero values should

contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v scp

Usage:

```
d8 v [command] [args...] scp [flags] SOURCE TARGET [options]
```

Secure file CoPy from/to a virtual machine.

Options

- `--identity-file`, `-i`

  Specify a path to a private key used for authenticating to the server; If not provided, the client will try to use the local ssh-agent at $SSH_AUTH_SOCK
- `--known-hosts`

  Set a path to the known_hosts file.
- `--local-ssh`

  Use the SSH/SCP client available on your system by using this command as ProxyCommand; Default is false: use embedded SSH client with limited capabilities
- `--local-ssh-opts`, `-t`

  Additional options to be passed to the ssh client if --local-ssh=true is set
- `--port`, `-p`

  Specify a port to connect to
- `--preserve`

  Preserves modification times, access times, and modes from the original file.
- `--recursive`, `-r`

  Recursively copy entire directories

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS

- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v ssh

Usage:

```
d8 v [command] [args...] ssh [-n|--namespace NAMESPACE] VIRTUAL-MACHINE-NAME [options]
```

Open a SSH connection to a virtual machine.

Options

- `--command`, `-c`

  Specify a command to execute in the VM.
- `--identity-file`, `-i`

  Specify a path to a private key used for authenticating to the server; If not provided, the client will try to use the local ssh-agent at $SSH_AUTH_SOCK

- `--known-hosts`

  Set a path to the known_hosts file.
- `--local-ssh`

  Use the SSH/SCP client available on your system by using this command as ProxyCommand; Default is false: use embedded SSH client with limited capabilities
- `--local-ssh-opts`, `-t`

  Additional options to be passed to the ssh client if --local-ssh=true is set
- `--port`, `-p`

  Specify a port to connect to
- `--username`, `-l`

  Specify user to log into virtual machine; If unassigned, this will be empty and the SSH default will apply

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.

- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v start

Usage:

```
d8 v [command] [args...] start (VirtualMachine) [options]
```

Start a virtual machine.

Options

- `--create-only`, `-c`

  Set this flag to only create the action without status warnings or notifications.
- `--force`, `-f`

  Set this flag to force the operation.
- `--timeout`, `-t`

  Set this flag to change the timeout.
- `--wait`, `-w`

  Set this flag to wait for the operation to complete.

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v stop

Usage:

```
d8 v [command] [args...] stop (VirtualMachine) [options]
```

Stop a virtual machine.

Options

- `--create-only`, `-c`

  Set this flag to only create the action without status warnings or notifications.
- `--force`, `-f`

  Set this flag to force the operation.
- `--timeout`, `-t`

  Set this flag to change the timeout.
- `--wait`, `-w`

  Set this flag to wait for the operation to complete.

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

  comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

d8 v vnc

Usage:

```
d8 v [command] [args...] vnc VirtualMachine [options]
```

Open a vnc connection to a virtual machine.

Options

- `--address`

  --address=127.0.0.1: Setting this will change the listening address of the VNC server. Example: --address=0.0.0.0 will make the server listen on all interfaces.
- `--port`

  --port=0: Assigning a port value to this will try to run the proxy on the given port if the port is accessible; If unassigned, the proxy will run on a random port

- `--proxy-only`

  --proxy-only=false: Setting this true will run only the vnc proxy and show the port where VNC viewers can connect

Global options

- `--as`

  Username to impersonate for the operation
- `--as-group`

  Group to impersonate for the operation, this flag can be repeated to specify multiple groups.
- `--as-uid`

  UID to impersonate for the operation
- `--certificate-authority`

  Path to a cert file for the certificate authority
- `--client-certificate`

  Path to a client certificate file for TLS
- `--client-key`

  Path to a client key file for TLS
- `--cluster`

  The name of the kubeconfig cluster to use
- `--context`

  The name of the kubeconfig context to use
- `--disable-compression`

  If true, opt-out of response compression for all requests to the server
- `--insecure-skip-tls-verify`

  If true, the server's certificate will not be checked for validity. This will make your HTTPS connections insecure
- `--kubeconfig`

  Path to the kubeconfig file to use for CLI requests.
- `--log-flush-frequency`

  Maximum number of seconds between log flushes
- `--namespace`, `-n`

  If present, the namespace scope for this CLI request
- `--password`

  Password for basic authentication to the API server
- `--proxy-url`

  If provided, this URL will be used to connect via proxy
- `--request-timeout`

  The length of time to wait before giving up on a single server request. Non-zero values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
- `--server`, `-s`

  The address and port of the Kubernetes API server
- `--tls-server-name`

  If provided, this name will be used to validate server certificate. If this is not provided, hostname used to contact the server is used.
- `--token`

  Bearer token for authentication to the API server
- `--user`

  The name of the kubeconfig user to use
- `--username`

  Username for basic authentication to the API server
- `--v`, `-v`

  number for the log level verbosity
- `--vmodule`

comma-separated list of pattern=N settings for file-filtered logging (only works for the default text log format)

Frequently Asked Questions

The page contains frequently asked questions about configuring and using the Deckhouse Kubernetes Platform.

Common

How can I apply an update for a specific module immediately?

To apply an update for a specific module immediately, set the `modules.deckhouse.io/apply-now: "true"` annotation on the corresponding ModuleRelease resource.

This annotation applies the release immediately without waiting for the update window. The requirements from `spec.requirements` still apply. If they are not met, the release will not be applied.

Example of setting the annotation for the `console` module:

```
d8 k annotate mr console-v1.43.3 modules.deckhouse.io/apply-now="true"
```

This can also be done using the `d8` CLI for convenience (module names and versions are autocompleted):

```
d8 system module apply-now console v1.43.3
```

Example resource with the annotation set:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleRelease
metadata:
  name: console-v1.43.3
  annotations:
    modules.deckhouse.io/apply-now: "true"
...
```

How can I apply an update DKP immediately, bypassing update windows, canary releases, and manual update mode?

To apply a Deckhouse Kubernetes Platform (DKP) update immediately, add the annotation `release.deckhouse.io/apply-now: "true"` to the corresponding DeckhouseRelease resource.

This will bypass update windows, canary release settings, and the manual cluster update mode. The update will be applied immediately after the annotation is set.

Example command to set the annotation and skip update windows for version `v1.56.2`:

```
d8 k annotate deckhousereleases v1.56.2 release.deckhouse.io/apply-now="true"
```

Example of a resource with the annotation set:

```
apiVersion: deckhouse.io/v1alpha1
kind: DeckhouseRelease
metadata:
  annotations:
    release.deckhouse.io/apply-now: "true"
...
```

What to do if, with VPN enabled, the container with the installer cannot access the network?

If a VPN is installed on the computer from which the cluster bootstrap is performed, there may be a problem with the container with the installer (or graphical installer) accessing the network. Because of this, for example, the graphical installer may not be displayed in the browser.

The problem can be solved in one of the following ways:

- Disable VPN on your computer and restart the container with the installer.
- If you cannot disable VPN (for example, if the cluster bootstrap is running on a VPN network), use the `--network host` parameter when starting the container with the installer (**for Docker Desktop on Mac OS, the parameter is available starting with version 4.34.0**). This will allow the container to access the network.

  Example of launching a container with an installer with VPN enabled, using the `--network host` parameter:

```
docker run --network host --pull=always -it -v "$PWD/config.yml:/config.yml" -v
"$HOME/.ssh/:/tmp/.ssh/" -v "$PWD/dhctl-tmp:/tmp/dhctl"
registry.deckhouse.ru/deckhouse/ce/install:early-access bash
```

Example of running the graphical installer with VPN enabled, using the `--network host` parameter:

```
docker run --network host --rm --pull always -v $HOME/.d8installer:$HOME/.d8installer -v
/var/run/docker.sock:/var/run/docker.sock -p 127.0.0.1:8080:8080
registry.deckhouse.ru/deckhouse/installer:latest -r $HOME/.d8installer
```

How do I change the release channel for a module?

A module can be built-in to DKP or connected from a module source (defined using ModuleSource). Built-in modules have a common release cycle with DKP and are updated together with DKP. **The release channel of a built-in module always matches the DKP release channel.** A module connected from a source has its own release cycle, which is independent of the DKP release cycle. **The release channel of a module connected from a source can be changed.**

Below is the process of changing the release channel for a module connected from a source.

By default, the release channel for modules is inherited from the DKP release channel (specified in the `releaseChannel` parameter of the `deckhouse` ModuleConfig). For more information about release channels, see the Release Channels section.

For modules from a source, the release channel is specified using ModuleUpdatePolicy, which is then *linked* to the module via the `updatePolicy` parameter in ModuleConfig.

To change the release channel for a module from a source, follow these steps:

1. Define the module update policy.

   Create a ModuleUpdatePolicy where you specify the release channel in the `releaseChannel` parameter.

   Example ModuleUpdatePolicy:

   ```
   apiVersion: deckhouse.io/v1alpha2
   kind: ModuleUpdatePolicy
   metadata:
     name: my-module-policy
   spec:
     releaseChannel: Alpha
     update:
       mode: Auto
   ```

   If necessary, change the update mode and configure update windows.

   Ensure that the policy has been created:

   ```
   d8 k get mup my-module-policy
   ```

   Output example:

   ```
   NAME               RELEASE CHANNEL   UPDATE MODE
   my-module-policy   Alpha             AutoPatch
   ```

2. Link the update policy to the module.

   Specify the name of the created update policy in the updatePolicy parameter of the corresponding module's ModuleConfig.

   To edit the ModuleConfig, use the command (specify the module name):

   ```
   d8 k edit mc my-module
   ```

   Example ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: my-module
spec:
  enabled: true
  # ModuleUpdatePolicy name.
  updatePolicy: my-module-policy
```

When you change the module's release channel, its version will change according to the configured update mode.

To view the current release channel of the module and other information about the module's state in the cluster, use the corresponding Module object.

Example command to get information about the module:

```
d8 k get module my-module -o yaml
```

The update policy used will be specified in the `properties.updatePolicy` field, and the current release channel in the `properties.releaseChannel` field. Example output:

```
apiVersion: deckhouse.io/v1alpha1
kind: Module
metadata:
  name: my-module
  # ...
properties:
  # ...
  releaseChannel: Alpha # Module release channel.
  updatePolicy: my-module-policy # Module update policy.
  version: v1.16.10  # Module version.
  # ...
```

How do I configure an external module to update to patch versions only?

By default, if a module has no dedicated update policy, the update mode and windows are inherited from the DKP settings.

If DKP is set to the `AutoPatch` mode, the external module will also automatically receive only patch versions within the current minor version. Moving to a new minor version will require manual approval. In this case, no additional configuration is needed.

For more information about DKP update modes, see Configuring updates.

If you need to manage the module update mode independently of DKP, create a ModuleUpdatePolicy with `update.mode: AutoPatch` and link it to the module via the `updatePolicy` parameter in ModuleConfig:

1. Create an update policy.

   Example ModuleUpdatePolicy:

   ```
   apiVersion: deckhouse.io/v1alpha2
   kind: ModuleUpdatePolicy
   metadata:
     name: my-update-policy
   spec:
     releaseChannel: Stable
     update:
       mode: AutoPatch
   ```

   Ensure that the policy has been created:

   ```
   d8 k get mup my-update-policy
   ```

2. Link the policy to the module.

   Specify the policy name in the updatePolicy parameter of the module's ModuleConfig:

   ```
   d8 k edit mc module
   ```

Example ModuleConfig:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: module
spec:
  enabled: true
  updatePolicy: my-update-policy
```

In the `AutoPatch` mode, module patch versions (for example, from `v1.16.1` to `v1.16.2`) are applied automatically, taking update windows into account if they are configured. To move to a new minor version (for example, from `v1.16.*` to `v1.17.*`), approve the corresponding ModuleRelease:

```
d8 k annotate mr module-v1.17.0 modules.deckhouse.io/approved="true"
```

Or using the `d8` CLI:

```
d8 system module approve module v1.17.0
```

What should I do if the module image did not download and the module did not reinstall?

In some cases, there may be a problem with automatically downloading the image and reinstalling the module. These cases include:

- Damage to the file system or other problems that have rendered the module image invalid.
- Switching to a different registry.
- Switching from one DKP edition to another.

In this case, the module may be in the `Ready` state. The error occurs in the module's pods. To find the problematic pod, use the command:

```
d8 k -n d8-<module-name> get pods
```

A problematic pod will have a status other than `Running`.

To view information about a pod, use the command:

```
d8 k -n d8-<module-name> describe pod <pod-name>
```

Example of an error message in the pod when there is a problem with downloading the image and reinstalling the module:

```
Failed to pull image
"registry.deckhouse.ru/deckhouse/ce/modules/console@sha256:a12b4f8de1d997005155d0ba0a7c968a0
15dd8d18bb5d54645ddb040ddab1ef4": rpc error: code = NotFound desc = failed to pull and
unpack image
"registry.deckhouse.ru/deckhouse/ce/modules/console@sha256:a12b4f8de1d997005155d0ba0a7c968a0
15dd8d18bb5d54645ddb040ddab1ef4": failed to resolve reference ...
```

To download the image and reinstall the module that caused the problem:

1. Get a list of module releases:

   ```
   d8 k get mr -l module=my-module
   ```

   Output example:

   ```
   NAME               PHASE        UPDATE POLICY   TRANSITIONTIME   MESSAGE
   my-module-v3.7.4   Superseded                   5d23h
   my-module-v3.7.5   Deployed                     5d23h
   ```

   Find the module release deployed in the cluster in the list (it should have the status `Deployed`).

2. Add the annotation `modules.deckhouse.io/reinstall=true` to the expanded release:

   ```
   d8 k annotate mr my-module-v3.7.5 modules.deckhouse.io/reinstall=true
   ```

After adding the annotation, the module image is re-downloaded from the registry, the module is validated with the current settings from `ModuleConfig`, and installed in the cluster. After successful reinstallation, the annotation is automatically removed from `ModuleRelease`.

To verify that the module has been successfully reinstalled and all module pods are working, use the command:

```
d8 k -n d8-<module-name> get pods
```

All pods in the module must have the status `Running`. Example:

```
NAME                                READY   STATUS    RESTARTS   AGE
backend-567d6c6cdc-g5qgt            1/1     Running   0          2d2h
frontend-7c8b567759-h8jdf           1/1     Running   0          2d2h
observability-gw-86cf75f5d6-7xljh   1/1     Running   0          2d2h
```

Cluster and Infrastructure

What to do if you encounter problems updating Kubernetes components on cluster nodes, synchronizing nodes, or applying NodeGroup Configuration?

If Kubernetes components are not updated on the cluster node, the NodeGroup configuration is not applied, and not all NodeGroup nodes are synchronized (have the `UPTODATE` status), perform the following steps:

1. Check the bashible logs on the node where the problems are occurring. The bashible mechanism is used to keep cluster nodes up to date. It is started by the `bashible.timer` timer at regular intervals as a service on the cluster nodes. This involves restarting, synchronizing scripts, and executing them (if necessary).

   To check bashible logs, use the command:

   ```
   journalctl -u bashible
   ```

   If the response contains the message `Configuration is in sync, nothing to do`, the node is synchronized and there are no problems. The absence of this message or the presence of errors indicates a problem.

2. Check the synchronization status of cluster nodes using the command:

   ```
   d8 k get ng
   ```

   The number of nodes in the `UPTODATE` state must match the total number of nodes in each group.

   Example output:

   ```
   NAME       TYPE     READY   NODES   UPTODATE   INSTANCES   DESIRED   MIN   MAX   STANDBY
     STATUS   AGE    SYNCED
   frontend   Static   1       1       1
              118d   True
   master     Static   3       3       3
              118d   True
   system     Static   2       2       2
              118d   True
   worker     Static   2       2       2
              118d   True
   ```

What to do if problems adding a node to the cluster via Cluster API Provider Static are present?

If, when adding a node to the cluster via Cluster API Provider Static (CAPS), it remains in `Pending` or `Bootstrapping` status, perform the following steps:

1. Verify that the access keys specified in the SSHCredentials resource are correct. Ensure that the username and SSH key specified in SSHCredentials are correct.
2. On the node where the problem occurred, check that the public key corresponding to the private key from SSHCredentials is present in `authorized_keys`. Example command for checking:

   ```
   cat ~/.ssh/authorized_keys
   ```

3. Check the number of nodes specified in NodeGroup, which should include the node being added. Make sure that the maximum number of nodes is not exceeded.

4. Check the status of the `bashible.service` on the node that caused the problem:

   ```
   systemctl status bashible.service
   ```

   It must have the status `active (running)`. If the service has the status `inactive` or `failed`, the service has not started. This indicates a problem with the configuration process.

5. If the steps above did not resolve the issue, remove the problematic node and its StaticInstance resource from the cluster so that the system will attempt to recreate them. To do this:

   - Get a list of nodes and locate the problematic one:

     ```
     d8 k get nodes
     ```

   - Find the corresponding StaticInstance resource:

     ```
     kubectl get staticinstances -n <namespace-name>
     ```

   - Remove the problematic node:

     ```
     kubectl delete node <node-name>
     ```

   - Remove the corresponding StaticInstance resource:

     ```
     kubectl delete staticinstances -n <namespace-name> <static-instance-name>
     ```

How to change the instance type for nodes with the CloudPermanent type?

To change the instance type for nodes with the CloudPermanent type, follow these steps:

1. Make a backup of etcd and the `/etc/kubernetes` directory.
2. Transfer the archive to a server outside the cluster (e.g., on a local machine).
3. Ensure there are no alerts in the cluster that can prevent the update of the master nodes.
4. Make sure that Deckhouse queue is empty. To view the status of all Deckhouse job queues, run the following command:

   ```
   d8 s queue list
   ```

   Example output (queues are empty):

   ```
   Summary:
   - 'main' queue: empty.
   - 88 other queues (0 active, 88 empty): 0 tasks.
   - no tasks to handle.
   ```

5. Run the appropriate edition and version of the Deckhouse installer container **on the local machine** (change the container registry address if necessary):

   ```
   DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
   DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]'
   '[:lower:]' )
   docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
     registry.deckhouse.ru/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
   ```

6. **In the installer container**, run the following command to check the state before working:

   ```
   dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
     --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
   ```

   The command output should indicate that Terraform found no inconsistencies and no changes are required.

7. **In the installer container**, run the command to edit the cluster configuration (specify the addresses of all master nodes in the `--ssh-host` parameter):

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME>
--ssh-user=<USERNAME> \
  --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

8. Edit the `instanceClass` parameter of the desired node group by changing the instance type and save the changes. Example settings for the `masterNodeGroup` of the Yandex Cloud provider:

   ```
   masterNodeGroup:
    replicas: 3  # required number of master nodes
    instanceClass:
      cores: 4     # change the number of CPUs
      memory: 8192  # change the memory size (in MB)
      # other instance parameters...
      externalIPAddresses:
      - "Auto"     # for each master node
      - "Auto"
      - "Auto"
   ```

9. **In the installer container**, run the following command to perform nodes upgrade:

   You should read carefully what converge is going to do when it asks for approval.

   When the command is executed, the nodes will be replaced by new nodes with confirmation on each node. The replacement will be performed one by one in reverse order (2,1,0).

   ```
   dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
     --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
   ```

   Repeat the steps below (Sec. 9-12) for each master node one by one, starting with the node with the highest number (suffix 2) and ending with the node with the lowest number (suffix 0).

10. **On the newly created node**, check the systemd-unit log for the `bashible.service`. Wait until the node configuration is complete (you will see a message `nothing to do` in the log):

    ```
    journalctl -fu bashible.service
    ```

11. Make sure the node is listed as an etcd cluster member:

    ```
    for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
      d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
      --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
      --endpoints https://127.0.0.1:2379/ member list -w table
      if [ $? -eq 0 ]; then
        break
      fi
    done
    ```

12. Make sure `control-plane-manager` is running on the node:

    ```
    d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
      -l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER-NODE-N-NAME>
    ```

13. Proceed to update the next node (repeat the steps above).

What to do if it takes a long time to switch to custom nodes in lower-priority groups?

When using multiple node groups with different priorities in a cloud cluster (the `spec.cloudInstances.priority` parameter), switching to nodes from groups with lower priority can take a long time. An example of this scenario is when groups of preemptible nodes (spot, etc.) are set to the highest priority, and if such instances are unavailable, ordering nodes from other groups takes a very long time.

When provisioning nodes, the cluster autoscaler sequentially selects the group with the highest priority that is **not in a backoff state**. A backoff is a temporary lock on a group following a failed attempt to provision a node (for example, due to a lack of instances of the required type in the cloud).

How it works when using multiple node groups:

1. The cluster autoscaler attempts to provision a node in the group with the highest priority.
2. If the node is not provisioned within the time specified by the `max-node-provision-time` parameter, the attempt is considered a failure.

3. The group is marked as `failed` and blocked for the duration specified in the `initial-node-group-backoff-duration` parameter.
4. If the same group fails again, the lockout time is doubled, but does not exceed the time specified in the `max-node-group-backoff-duration` parameter.
5. The cluster autoscaler selects the next highest-priority group that is not locked out.

Here:

Parameter	Default value	Description
`initial-node-group-backoff-duration`	5 minutes	Initial duration of the group lockout after the first failure. Doubles after each failed node ordering attempt
`max-node-group-backoff-duration`	30 minutes	Maximum duration of the group lockout
`max-node-provision-time`	15 minutes	The time after which the cluster autoscaler considers a node provisioning attempt to have failed

If these parameters are set to their default values, switching to node ordering in lower-priority groups may take a considerable amount of time (for more details, see this example).

These settings can be adjusted to speed up the switching of nodes in lower-priority groups. To set the desired values for the `initial-node-group-backoff-duration`, `max-node-group-backoff-duration`, and `max-node-provision-time` parameters, edit the `cluster-autoscaler` Deployment object in the `d8-cloud-instance-manager` namespace. The parameters are specified in the `args` field of the `cluster-autoscaler` container. Example:

```
...
    containers:
     - name: cluster-autoscaler
       args:
       - --initial-node-group-backoff-duration=1m # The initial duration of the group lockout
after the first failure has been reduced.
       - --max-node-provision-time=5m # The time after which the cluster autoscaler considers
a node request to have failed has been reduced.
...
```

If you encounter issues with node ordering in a cluster with a single node group

If cloud cluster uses only one node group (for example, exclusively for ordering spot or preemptible nodes) and you encounter issues with ordering such nodes during automatic scaling, follow these steps:

1. Create one or more InstanceClass objects to be used when creating instances in the cluster. In the InstanceClass, specify node types that differ from those listed in the group mentioned above.

   > **i** In DKP, InstanceClass objects vary depending on the provider. Examples: AWSInstanceClass, AzureInstanceClass, YandexInstanceClass.

2. Create one or more new node groups (NodeGroup). In the `spec.cloudInstances.classReference` parameter, specify the InstanceClass objects created in the previous step. In the `spec.cloudInstances.priority` parameter, set the priority of the node group.
3. If necessary, modify the `max-node-provision-time`, `max-node-group-backoff-duration`, and `initial-node-group-backoff-duration` parameters to speed up switching to lower-priority groups.

Example of the node ordering process with default settings

Suppose there are 3 groups of nodes with different priorities:

- **Group A** (priority 50)
- **Group B** (priority 30)
- **Group C** (priority 0)

In this case, the node ordering process might be as follows:

Time	Event
10:00	The cluster autoscaler attempts to order a node in **A**
10:15	The attempt to provision a node in **A** within 15 minutes (`max-node-provision-time`) fails → **A** blocked for 5 minutes (`initial-node-group-backoff-duration`)

Time	Event
10:15:xx	The cluster autoscaler attempts to order a node in **B** (next in priority, not blocked)
10:30	The attempt to provision a node in **B** within 15 minutes (`max-node-provision-time`) fails → **B** blocked for 5 minutes (`initial-node-group-backoff-duration`)
10:30:xx	The cluster autoscaler attempts to order a node in **A** again, as the lockout period (the default value for `initial-node-group-backoff-duration` is 5 minutes) has ended
10:45	The attempt to order a node in **A** fails again → **A** is blocked for 10 minutes (`initial-node-group-backoff-duration` for **A** is doubled)
…	And so on, until the lockout time for **A** and **B** exceeds 15 minutes (`max-node-provision-time`). In this case, the queue may not reach group **C** for about 1.5 hours

What to do if the resources generated by an object are not created after adding the object?

If the required resources do not appear after creating an object in the system (for example, dexAuthenticator), follow these steps:

1. Check if there are any critical alerts in the cluster that may block the creation of the required objects. To do this, use the command:

   ```
   d8 k get clusteralerts.deckhouse.io
   ```

   Example output:

   ```
   NAME               ALERT                                        SEVERITY   AGE   LAST
   RECEIVED   STATUS
   012f602592aa7a91   K8SSchedulerTargetDown                       3          16h   54s
   firing
   0836dc893d5ecc65   KubernetesDeploymentReplicasUnavailable      5          15h   62s
   firing
   08742f87d62d0063   NTPDaemonOnNodeDoesNotSynchronizeTime        5          16h   46s
   firing
   172cfd38d2f7fd19   D8DeckhouseQueueIsHung                       7          12h   66s
   firing
   1c5705daf731f5cf   D8StrongholdNoActiveNodes                    3          16h   55s
   firing
   1d2c2f7d69f69f4b   D8DeckhouseIsNotOnReleaseChannel             9          12h   53s
   firing
   205a551243d795f3   D8LogShipperAgentNotScheduledInCluster       7          15h   63s
   firing
   2e34039aa7a3018e   D8NodeIsNotUpdating                          9          12h   47s
   firing
   31baf9a70d657275   D8StrongholdClusterNotHealthy                7          16h   55s
   firing
   ```

   For more information about alerts, see the The list of alerts section.

2. Check the Deckhouse task queue:

   ```
   d8 s queue list
   ```

   Example output (queues are empty):

   ```
   Summary:
   - 'main' queue: empty.
   - 88 other queues (0 active, 88 empty): 0 tasks.
   - no tasks to handle.
   ```

   If there are many unprocessed or long-running tasks in the queue, this may indicate problems.

3. Analyze DKP logs and events:
   - To view logs in real time, use the command:

     ```
     d8 k -n d8-system logs -f -l app=deckhouse
     ```

     Example output:

     ```
     {"level":"info","logger":"addon-operator","msg":"ConvergeModules task for
     OperatorStartup in phase '', trigger is Operator-
     Startup","binding":"ConvergeModules","event.type":"OperatorStartup","queue":"main",
     "task.flow":"start","task.id":"fde0eb3b-5c3e-4da6-a0d8-a52f8ae03428","time":"2025-
     11-26T08:29:33Z"}
     {"level":"warn","logger":"addon-operator.converge-modules","msg":"ConvergeModules:
     functional scheduler not
     finished","binding":"ConvergeModules","event.type":"OperatorStartup","queue":"main"
     ,"task.id":"fde0eb3b-5c3e-4da6-a0d8-a52f8ae03428","time":"2025-11-26T08:29:33Z"}
     ```

     When analyzing logs, pay special attention to warnings (`WARNING`) and error messages (`ERROR`).
   - To view events, use the command:

     ```
     d8 k -n d8-system get events
     ```

     Example output:

     ```
     LAST SEEN   TYPE      REASON              OBJECT
     MESSAGE
     11m         Warning   Unhealthy           pod/deckhouse-5886c9bd77-vgdbw
     Readiness probe failed: HTTP probe failed with statuscode: 500
     7m22s       Normal    SuccessfulDelete    replicaset/deckhouse-5886c9bd77
     Deleted pod: deckhouse-5886c9bd77-vgdbw
     7m20s       Normal    Scheduled           pod/deckhouse-6bc5c4494-fwx6z
     Successfully assigned d8-system/deckhouse-6bc5c4494-fwx6z to sandbox1-master-0
     7m20s       Normal    Pulling             pod/deckhouse-6bc5c4494-fwx6z
     Pulling image "dev-registry.deckhouse.io/sys/deckhouse-
     oss@sha256:17ac07634e17422df52720264cddec3916ed6985a77782dc8a24fe5352290e6e"
     ```

   When analyzing events, pay special attention to those with the `Warning` type.

Deckhouse

How can I check DKP versions available in different release channels?

For information about the current DKP versions across all release channels, visit releases.deckhouse.io.

How to check the job queue in Deckhouse?

How to check the status of all Deckhouse task queues?

To view the status of all Deckhouse job queues, run the following command:

```
d8 s queue list
```

Example output (queues are empty):

```
Summary:
- 'main' queue: empty.
- 88 other queues (0 active, 88 empty): 0 tasks.
- no tasks to handle.
```

How to view the status of the main task queue?

To view the status of the Deckhouse `main` task queue, run the following command:

```
d8 s queue main
```

Example output (38 tasks in the `main` queue):

```
Queue 'main': length 38, status: 'run first task'
```

Example output (the `main` queue is empty):

```
Queue 'main': length 0, status: 'waiting for task 0s'
```

What should I do if DKP is not receiving updates from the configured channel?

- Ensure the correct release channel is configured.
- Check that DNS resolution for the Deckhouse image registry is working correctly.

  Get and compare the IP addresses of `registry.deckhouse.io` from both a node and the `deckhouse` Pod. They must match.

  Example of obtaining an IP of `registry.deckhouse.io` from a node:

  ```
  getent ahosts registry.deckhouse.io
  ```

  Example output:

  ```
  185.193.90.38    STREAM registry.deckhouse.io
  185.193.90.38    DGRAM
  185.193.90.38    RAW
  ```

  Example of obtaining an IP of `registry.deckhouse.io` from the `deckhouse` Pod:

  ```
  d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- getent ahosts registry.deckhouse.io
  ```

  Example output:

  ```
  185.193.90.38    STREAM registry.deckhouse.io
  185.193.90.38    DGRAM  registry.deckhouse.io
  ```

  If the resulted IPs do not match, check DNS settings on the node. Pay attention to the `search` domain list in `/etc/resolv.conf`, which affects name resolution in the `deckhouse` Pod. If the `search` parameter in `/etc/resolv.conf` file specifies a domain with wildcard DNS resolution configured, this may lead to incorrect IP address resolution for the Deckhouse image registry (see example below).

Example DNS settings that may cause issues resolving the DKP image registry IP address

Below is an example of how DNS settings may result in different resolution behavior on the node and in a Kubernetes Pod:

- Example of `/etc/resolv.conf` on the node:

  ```
  nameserver 10.0.0.10
  search company.my
  ```

  > **i** On nodes, the default `ndot` setting is **1** (`options ndots:1`), while in Kubernetes Pods, it's **5**. This causes different resolution logic for DNS names with 5 or fewer dots on a node and on the Pod.

- The DNS zone `company.my` has a wildcard entry `*.company.my` that resolves to `10.0.0.100`. This means any undefined DNS name in the `company.my` zone resolves to `10.0.0.100`.

Taking into account the `search` parameter in `/etc/resolv.conf`, when accessing `registry.deckhouse.io` from a node, the system will attempt to resolve the IP address for `registry.deckhouse.io` (because it considers it fully qualified due to the default `options ndots:1` setting).

However, when accessing `registry.deckhouse.io` from a Kubernetes Pod, considering the `options ndots:5` setting used by default in Kubernetes and the `search` parameter, the system will first attempt to resolve the name `registry.deckhouse.io.company.my`. This name will resolve to the IP address `10.0.0.100` because, according to the `company.my` DNS zone's wildcard configuration, `*.company.my` is resolved to `10.0.0.100`. As a result, the Pod will fail to connect to the `registry.deckhouse.io` host and will be unable to download information about available Deckhouse updates.

Does Deckhouse support realtime (rt) and low-latency Linux kernels?

In general, realtime (rt) and lowlatency kernels are supported with no extra configuration needed for Deckhouse Kubernetes Platform. Such kernels have been tested with DKP EE version v1.75.7 and above on the following

OSes and kernels:

- CentOS 9 Stream, linux-5.14.0-706.el9.x86_64+rt
- RedOS 8, linux-6.12.85-1.red80.x86_64-rt
- Astra Linux 1.7.5, linux-5.15-lowlatency

How can I know when a new DKP version is available for the cluster?

As soon as a new version appears on the configured release channel:

- The `DeckhouseReleaseIsWaitingManualApproval` alert will appear if the cluster is in manual update mode.
- A new DeckhouseRelease custom resource will be created. To see the list of releases, run `d8 k get deckhousereleases`. If the new version is in `Pending` state, it means it hasn't been installed yet. Possible reasons:
  - Manual update mode is enabled.
  - Automatic update mode is enabled and update windows are scheduled, but the window hasn't started yet.
  - Automatic update mode is enabled and update windows are not scheduled, but the update is delayed by a random period to reduce load on the container image registry. The `status.message` field of the DeckhouseRelease resource will show a corresponding message.
  - The `update.notification.minimalNotificationTime` parameter is set, and the delay period hasn't elapsed.

How can I receive information about upcoming updates in advance?

You can get information about upcoming minor DKP version updates on the release channel in one of the following ways:

- Enable manual update mode. A new DeckhouseRelease resource and the `DeckhouseReleaseIsWaitingManualApproval` alert will appear when a new version is available.
- Enable automatic update mode and set a delay using the `minimalNotificationTime` parameter. A new DeckhouseRelease resource will appear when a new version is available. If you also set a webhook URL in the `update.notification.webhook` parameter, a notification will be sent about the upcoming update.

How can I tell that the update was successful?

If the `DeckhouseUpdating` alert is gone, the update has finished.

You can also check the status of DKP releases in the cluster with the following command:

```
d8 k get deckhouserelease
```

Example output:

```
NAME       PHASE        TRANSITIONTIME   MESSAGE
v1.46.8    Superseded   13d
v1.46.9    Superseded   11d
v1.47.0    Superseded   4h12m
v1.47.1    Deployed     4h12m
```

The `Deployed` status means the cluster has switched to the corresponding version, but it doesn't guarantee that the update has been successful.

To ensure the update completed successfully, check the state of the `deckhouse` Pod with the following command:

```
d8 k -n d8-system get pods -l app=deckhouse
```

Example output:

```
NAME                   READY  STATUS   RESTARTS  AGE
deckhouse-7844b47bcd-qtbx9  1/1   Running  0       1d
```

- If the Pod is `Running` and shows `1/1` under `READY`, it means the update completed successfully.
- If the Pod is `Running` but shows `0/1` under `READY`, it means the update is still in progress. If it stays like this for more than 20–30 minutes, it may indicate a problem with DKP that requires investigation.
- If the Pod is not `Running`, it may indicate a problem with DKP that requires investigation.

If something goes wrong

- Check the logs using the following command:

```
d8 k -n d8-system logs -f -l app=deckhouse | jq -Rr 'fromjson? | .msg'
```

- Collect debug information and contact DKP technical support.

How can I tell that an update is in progress?

During an update:

- The `DeckhouseUpdating` alert is active.
- The `deckhouse` Pod is not in the `Ready` state. If the Pod stays in a non-`Ready` state for a long time, it may indicate an issue with DKP that requires investigation.

What to do if there are problems updating DKP?

Deckhouse Kubernetes Platform update fails, one or more Deckhouse pods are in an unworkable state

If the Deckhouse Kubernetes Platform update fails, one or more Deckhouse pods in the `d8-system` namespace are in an unworkable state. Perform the following steps:

1. Check the Deckhouse logs using the command:

   ```
   d8 k -n d8-system logs -f -l app=deckhouse | jq -Rr 'fromjson? | .msg'
   ```

   If there are any problems, information about them will be included in the output. When analyzing logs, pay special attention to warnings (`WARNING`) and error messages (`ERROR`).

2. Check Deckhouse events using the command:

   ```
   d8 k -n d8-system describe po -l app=deckhouse | awk '
   /^Name:/ {
       pod = $2;
       print "=== " pod " ===";
       in_events = 0
   }
   /Events:/ {
       in_events = 1;
       next
   }
   in_events && /^$/ {
       in_events = 0;
       print "---"
   }
   in_events && !/^Events:/ {
       print $0
   }
   ' | sed '/^---$/N;/^\n$/D'
   ```

   Pod events contain key information about problems (e.g., planning errors, image loading errors, etc.). When analyzing events, pay special attention to those with the `Warning` type.

   Example output:

   ```
   Type     Reason     Age                     From     Message
   ----     ------     ----                    ----     -------
   Warning  Unhealthy  4m44s (x1918 over 154m)  kubelet  Readiness probe failed: HTTP probe
   failed with statuscode: 500
   ```

DKP update is stuck in the Release is suspended status

The status `Release is suspended` indicates that it has been postponed and is currently unavailable (not recommended) for installation. In this case, it is recommended to remain on the latest available release or on the one currently installed (it will have the status `Deployed`).

To view the list of releases, use the command:

```
d8 k get deckhousereleases.deckhouse.io
```

Example output:

```
NAME       PHASE        TRANSITIONTIME   MESSAGE
v1.69.13   Skipped      3h46m
v1.69.14   Skipped      3h46m
v1.69.15   Skipped      3h46m
v1.69.16   Superseded   160m
v1.70.12   Suspended    49d              Release is suspended
v1.70.13   Skipped      36d
v1.70.14   Skipped      34d
v1.70.15   Skipped      28d
v1.70.16   Skipped      19d
v1.70.17   Deployed     160m
v1.71.3    Suspended    14d              Release is suspended
```

IAM

What to do if there are problems applying DexProvider settings?

If you have changed the DexProvider settings in the `user-authn` module and one of the following problems occurs:

- No changes are visible (settings are not applied).
- When attempting to log in to the platform's web interface with any type of authorization, a `500 Internal Server Error` occurs without a detailed description.

Perform the following steps:

1. Check the status of the dex deployment:

   ```
   d8 k -n d8-user-authn get pod
   ```

   Example output:

   ```
   NAME                                READY   STATUS    RESTARTS   AGE
   dex-5ddb779b7d-6pbhs                2/2     Running   0          20h
   kubeconfig-generator-7c46977b9f-5kdmc   1/1     Running   0          20h
   ```

   If the module is functioning properly and the correct configuration is specified in DexProvider, all pods will have the status `Running`.

2. Check the logs for the problematic pod:

   ```
   d8 k -n d8-user-authn logs dex-<pod-name>
   ```

   Based on the information from the logs, correct the configuration in the DexProvider resource and wait for the dex pods to restart. Within a few minutes, the pods will restart automatically, and the platform's web interface (located at `console.<CLUSTER_NAME_TEMPLATE>`) will become available and will reflect the changes made to the DexProvider resource.

Kubernetes and Scheduling

What to do when the API server is overloaded?

The following signs may indicate problems with API server load and memory consumption:

- `kubectl (d8)` responds slowly or does not respond at all (commands are executed slowly or not at all).
- Pods are recreated in the cluster for no apparent reason.

If these signs are present, perform the following actions:

1. Check the resource consumption of API server pods. To do this, use the command:

   ```
   d8 k -n kube-system top po -l component=kube-apiserver
   ```

   Pay attention to `MEMORY` consumption and `CPU`.

   Example output:

   ```
   NAME                              CPU(cores)   MEMORY(bytes)
   kube-apiserver-sandbox1-master-0   251m         1476Mi
   ```

2. Check the metrics in Grafana.

To view the metrics, open the dashboard “Home” → “Dashboards” → “Kubernetes Cluster” → “Control Plane Status”. Review the graphs related to the API server (“Kube-apiserver CPU Usage”, “Kube-apiserver Memory Usage”, “Kube-apiserver latency”, etc.).

3. Review the API server audit logs to identify the source of high memory consumption. One common cause of high memory consumption is a large number of requests.

How to check the Kubernetes version in use?

To check the Kubernetes version in use, run the command:

```
d8 k get nodes
```

Example output:

```
NAME                 STATUS   ROLES                  AGE    VERSION
frontend-0           Ready    frontend               118d   v1.31.9
master-0             Ready    control-plane,master   118d   v1.31.9
master-1             Ready    control-plane,master   118d   v1.31.9
master-2             Ready    control-plane,master   118d   v1.31.9
system-0             Ready    system                 118d   v1.31.9
system-1             Ready    system                 118d   v1.31.9
worker-0             Ready    worker                 37d    v1.31.9
worker-1             Ready    worker                 19d    v1.31.9
```

Network

Is the ingress-nginx module supported? Will it affect passing a PCI DSS audit?

The `ingress-nginx` module remains supported by Deckhouse Kubernetes Platform for the entire platform support lifecycle and does not depend on the upstream project’s status. The Deckhouse team tracks vulnerabilities in the controller and related components (NGINX, Lua modules, base images), delivers fixes in DKP releases, and provides a migration path toward Gateway API.

Flant is the vendor responsible for support. Together with ongoing module maintenance and vulnerability management, this allows passing a PCI DSS audit and meets expectations for vendor accountability for a platform component.

Release notes for Deckhouse Kubernetes Platform

Version 1.76

Important

- The fencing-agent component of the `node-manager` module now uses a gossip-based protocol (memberlist library) for distributed node health monitoring. This reduces the risk of false worker node reboots when the control plane is unavailable or the API server is under heavy load.
- Fixed an issue where `cluster-autoscaler` could get stuck after a failed machine creation in the cloud (for errors other than `ResourceExhausted`). The cluster now automatically recovers its ability to scale after temporary cloud provider failures.

Deckhouse subsystem

- Added support for deploying a DKP cluster using the `Proxy` mode for accessing the container registry.
- `Kubectl` on cluster nodes has been replaced with the `d8 k` alias (Deckhouse CLI utility).
- Changes in `dhctl` (CLI installer):
  - Added support for running `dhctl` as a standalone binary without downloading the full image with all dependencies. Required components are pulled from the registry during execution, reducing startup time.
  - Added validation of public SSH keys. Invalid keys are now detected earlier, before they can cause connection issues.
  - Improved handling of TLS certificates for registries. The system certificate pool is now used together with a custom CA.
  - Added validation of InstanceClass against the selected cloud provider before installation, allowing early detection and installation abort in case of misconfiguration.
  - Fixed namespace configuration during bootstrap. `Dhctl` now updates settings even if the namespace already exists.

- Improved multi-master resiliency. Kubernetes versions are now correctly detected even when some master nodes are unavailable. Kube-apiserver no longer routes requests to etcd nodes in the learner state.

Kubernetes & Scheduling subsystem

- The `d8-cluster-kubernetes` ConfigMap now includes supported Kubernetes versions (`supportedVersions`), versions available for upgrade and downgrade (`availableVersions`), the current automatic version (`automaticVersion`).
- Changes in the `descheduler` module:
  - Upgraded to version 0.35.1. Added namespace filtering via label selectors and other improvements (see the module changelog for details).
  - Added the `deschedulingInterval` parameter to control how often descheduling runs: `Frequent`, `Moderate`, or `Rare`.
  - Added the `RemovePodsHavingTooManyRestarts` strategy, which evicts pods exceeding a restart threshold, freeing node resources from pods in `CrashLoopBackOff` state.
  - The module now automatically uses the Kubernetes Metrics API when the `metrics.k8s.io` API group is available in the cluster. This provides real resource consumption data for strategies to use.
- The `vertical-pod-autoscaler` module updated to version 1.6.0. Key changes:
  - The `InPlaceOrRecreate` mode is now used by default. In this mode, VPA attempts to adjust pod requests and limits without recreating pods (in-place updates).
  - Reduced likelihood of pod restarts when in-place updates are not possible (via the `--in-place-skip-disruption-budget` flag), also `minReplicas` value is no longer considered for in-place updates.
- VPA memory recommendations are now rounded to 64 MiB, improving readability and reducing unnecessary pod restarts caused by minor recommendation fluctuations.
- For Kubernetes 1.34 and later, the `DRAExtendedResource` feature gate is enabled by default.
- Added the `D8EtcdHighRequestsLatency` alert to detect high etcd request latency, which can lead to API server timeouts. A corresponding metric has been added to the “Control Plane Status” dashboard.

IAM subsystem

- Changes in the `user-authn` module:
  - Disabled the insecure OAuth 2.0 Implicit Flow. In that flow, access tokens are passed in URL fragments, posing leakage risks (for example, via the `Referer` header, browser history, or logs). Any integrations using the Authorization Code Flow are unaffected.
  - Added SAML 2.0 support via the DexProvider resource. DKP clusters can now be connected to such identity providers as AD FS, Okta, Keycloak, OneLogin, and Shibboleth.
  - Added refresh token support for SAML in Dex. This allows DexAuthenticator и kubeconfig-generator to refresh tokens without a manual user login.
  - Added Single Logout (SLO) support. The identity provider can end user sessions via `/saml/slo/{connector}`.
  - Fixed stale cache issue on the login page by adding a Dex build-based `cache` parameter to the CSS URL.
- In multitenancy mode, users without permissions defined via ClusterAuthorizationRule no longer see all namespaces in the Deckhouse web UI. The default behavior is now deny-by-default: users get an empty namespace list unless access is explicitly granted. This does not affect privileged groups (`system:masters`, `kubeadm:cluster-admins`). Documentation of the `user-authz` module now includes an example for granting access to all namespaces.
- Added support for the `service-with-health-checks` module in isolated projects. A NetworkPolicy exception for the `d8-service-with-healthchecks` namespace has been added to the project template, which allows module agents to verify service availability and correctly operate under conditions of network isolation.

Security subsystem

- Gatekeeper upgraded to 3.22.0 and Ratify to 1.4.0. Key changes include a unified CEL and Rego policy behavior, added namespace context support in policies, and a CLI tool for policy benchmarking.
- The `denyVulnerableImages` setting has been moved from `admission-policy-engine` to `operator-trivy`, making it the single owner of this logic.
- The `cert-manager` module upgraded to 1.20.0. Key changes include added support for Azure DNS Private Zones in the DNS01 challenge, added annotation `acme.cert-manager.io/http01-ingress-ingressclassname` for redefining IngressClasses in HTTP-01 solvers, and security fixes.
- Updated vulnerable Go dependencies in `multitenancy-manager` (`go-jose`, `golang.org/x/crypto`, `grpc`,

etc.).

Cluster & Infrastructure subsystem

- Changes in the `node-manager` module:
  - Improved draining hook event generation for better observability of the node eviction process.
  - Fixed StaticInstance issue causing infinite infrastructure deletion loops. That prevents infinite cleanup and queueing cycles.
  - Apiserver-proxy rewritten from NGINX to native Go implementation for better stability and reconnection speed.
- Changes in the `cloud-provider-dvp` module:
  - Added hybrid cluster support. Static master nodes and cloud worker nodes can now be used in a single cluster.
  - Improved converge reliability. Reworked the mechanism of resource readiness check and added the fail-fast behavior for possible errors when creating the infrastructure.
  - Added default StorageClass inheritance for Cluster API child clusters.
  - Improved VM lifecycle handling, reducing potential problems when deleting VMs and associated objects.
  - Added support for the `masterNodeGroup.instanceClass.additionalDisks` parameter that lets you configure additional disks for master nodes.
- Changes in the `cloud-provider-aws` module:
  - Spot node draining logic was moved into DKP, making the spot node handling more unified.
  - Fixed handling of regions without `DescribeInstanceTopology`, reducing the number of false IAM errors.
- Changes in the `cloud-provider-openstack` module:
  - Added `ipMode: Proxy` support for LoadBalancer in Kubernetes 1.32 and later.
  - Added the `csiDriver.fsGroupPolicy` parameter for the CSI driver, which defines whether the volume supports changing owner and permissions before mounting.
- Changes in the `cloud-provider-vsphere` module:
  - Enabled the CSI snapshotter, enabling volume snapshotting.
  - Fixed zone and datastore detection. Now the only considered zones are the ones specified in the provider configuration.
- In the `cloud-provider-azure` module, added support for NVMe disks on Gen2 VMs.
- In the `cloud-provider-gcp` module, added `enableNestedVirtualization` and `additionalDisks` parameters to GCPInstanceClass, which let you configure nodes for workloads using virtualization and storage scenarios.
- In the `cloud-provider-yandex` module, fixed converge failure when removing externalIPAddresses.
- DVP, Huawei Cloud, and zVirt providers were migrated to Cluster API `v1beta2`.
- For Huawei Cloud and VCD providers, added SecurityPolicyException resources and enabled security policy validations.

Network subsystem

- Added support for handling incoming traffic via Gateway API using the `alb` module.
- Changes in the `cni-cilium` module:
  - Added conntrack table synchronization for live VM migration.
  - Added ICMP (ping) support via ExternalIP for LoadBalancer services with MetalLB.
- Added support for switching between supported CNI plugins in DKP clusters.
- In the `node-local-dns` module, added `disableIPv6` parameter to disable IPv6 DNS resolution.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

Version 1.75

Important

- Support for Kubernetes 1.35 has been added, while support for Kubernetes 1.30 has been discontinued. In future DKP releases, support for Kubernetes 1.31 will be removed. The default Kubernetes version (used when the `kubernetesVersion` parameter is set to `Automatic`) has been changed to 1.33.
- Update to DKP 1.75 will be blocked if the cluster uses one of the following deprecated component versions (they are no longer supported):
  - IngressNginxController v1.9
  - Istio v1.19
- In DKP 1.75.0–1.75.3, the `PodSecurityStandardsViolation` alert may fire immediately after cluster installation. This happens when the `d8-system` namespace is missing the `heritage=deckhouse` label, causing Pod Security Standards policies to be mistakenly applied to DKP system pods. The issue is fixed in DKP 1.75.4.

  For clusters installed on DKP 1.75.0–1.75.3, add the required label manually using the following command:

  ```
  d8 k --as=system:sudouser label namespace d8-system heritage=deckhouse
  ```
- Istio version support changes:
  - Support for Istio 1.19 has been discontinued.
  - Istio 1.21 is now considered deprecated, and support for this version will be discontinued in the upcoming DKP releases. To update Istio, follow the instruction.
- Starting with Kubernetes 1.35, cluster nodes must use cgroup v2. The previous version (cgroup v1) is considered deprecated. If cgroup v2 is not supported on a node, the D8NodeCgroupV2NotSupported alert will be triggered. See the Kubernetes documentation for cgroup v2 migration details.
- Starting with Kubernetes 1.36, cluster nodes must use containerd v2 as the container runtime. Support for containerd v1 will be discontinued. If containerd v2 is not supported on a node, the D8NodeContainerdV2NotSupported alert will be triggered.
- The `vertical-pod-autoscaler` module has been updated from version 1.4.1 to 1.5.1. Starting from version 1.5.0, the `updateMode: Auto` mode is considered deprecated. Users are advised to replace it with `updateMode: InPlaceOrRecreate`. See the vertical scaling documentation for details.
- When issuing new certificates using the `cert-manager` module, the certificate's private key is now re-generated as well. This change mitigates potential security risks.
- Pods handling HTTP-01 challenges via Issuer and ClusterIssuer resources now have default resource requests defined. As a result, these pods consume part of the cluster and project resource quotas upon creation.
- The operator-trivy module is now delivered from an external source and follows an independent release cycle. No manual migration steps are required.

Deckhouse subsystem

- The ModuleSource resource now includes the `scanInterval` parameter to control how frequently the container registry is checked for new modules and versions.
- A dedicated node mode with the `node.deckhouse.io/etcd-arbiter` label has been introduced. In this mode, only etcd runs on the node (no control plane components). This enables a highly available cluster with two master nodes plus a dedicated etcd node, reducing resource requirements compared to the traditional three full master nodes configuration. See the High Availability documentation for details.
- New operating modes have been introduced in the `registry` module: **Proxy** (with image caching) and **Local** (for isolated environments). These modes are currently supported only for already deployed static clusters and are not yet available during DKP cluster installation.
- Container registry access parameters can now be configured via the deckhouse module during DKP cluster installation. Support for the DKP cluster deployment via Direct and Unmanaged modes has been added. The Legacy Unmanaged mode (via InitConfiguration) is preserved for compatibility. If no explicit configuration is provided, the mode is selected automatically.
- Added user shell validation: if a shell other than bash is used, the `dhctl` utility now displays a clear error message.
- Added StaticInstance validation before DKP installation using the corresponding SSHCredentials, preventing `dhctl` from reporting readiness before the cluster deployment is actually complete.
- Extended preflight test coverage to improve cluster deployment predictability.
- Partially reduced usage of netcat in bashible, lowering the number of security system triggers.
- Fixed an issue where cluster installation could not be resumed after interruption.
- Fixed Node object deletion logic during `converge`. The node is now properly removed from the cluster after the corresponding VM is destroyed.
- Fixed the `--skip-resources` flag behavior in `dhctl destroy` command. Specified resources are now correctly skipped.

Kubernetes & Scheduling subsystem

- A new `update-observer` component in the `control-plane-manager` module enables real-time tracking of control plane component updates (kube-apiserver, kube-scheduler, kube-controller-manager) and kubelet on nodes. This helps detect discrepancies and eliminates “blind spots” during Kubernetes updates.
- For all VPA resources managed by DKP, `updateMode: Auto` has been automatically replaced with `updateMode: InPlaceOrRecreate`. For clusters running Kubernetes 1.32 or earlier, the `InPlacePodVerticalScaling` feature gate is automatically enabled in control plane components (kube-apiserver, kube-scheduler, kube-controller-manager) and kubelet. This allows pod resource changes without recreation.
- The descheduler module has been updated to version 0.34. Key changes:
  - Priority-based pod eviction: pods with a higher restart count are evicted first, improving cluster stability during load redistribution.
  - DRA support: proper handling of resources managed via Dynamic Resource Allocation.
  - Default `cpu`, `memory`, and `pods` thresholds removed from `lowNodeUtilization` and `highNodeUtilization` strategies to align with upstream behavior.

IAM subsystem

- Namespace listing behavior has been changed. When requesting all cluster namespaces, users now see only the namespaces they have access to, even without permission to list all namespaces. This improves working with multitenancy and UI.
- The ProjectTemplate resource in the `multitenancy-manager` module now supports the unmanaged and skip-heritage options (set via annotations), allowing administrators to adjust the DKP control degree over resources:
  - `unmanaged`: The module does not manage the lifecycle of objects.
  - `skip-heritage`: DKP retains management but allows modification by external controllers.
- Added support for authenticating to the Kubernetes API via LDAP using Basic authentication instead of OIDC tokens (via the `ldap.enableBasicAuth` parameter in the DexProvider resource).
- The authorization webhook in the `user-authz` module now operates in fail-closed mode: if it’s unavailable, authorization is denied. For improved reliability, webhook communication with the apiserver is performed via the cluster node network.
- The `user-authn` module now supports a global user session lifetime limit (via `refreshTokenAbsoluteLifetime`). Re-authentication is required after expiration.
- The `user-authn` module now supports Kerberos (SPNEGO) authentication in DKP.
- Updated kube-apiserver TLS cipher suites. Removed the following insecure ciphers from the `--tls-cipher-suites` flag: TLS_RSA_WITH_AES_256_GCM_SHA384 and TLS_RSA_WITH_AES_128_GCM_SHA256 (considered unsafe due to RSA key exchange). Added TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 and TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256.
- The anonymous access restriction mechanism for the API server has been modified. Instead of using the `--anonymous-auth=false` flag and a sidecar proxy, the AuthenticationConfiguration resource is now used to allow anonymous access only to `/livez`, `/readyz`, and `/healthz`. This improves cluster resource efficiency. Anonymous probe access remains restricted when public API access is enabled via `publishAPI.enabled`.

Security subsystem

- Fixed serialization of empty arrays and objects in SecurityPolicy and OperationPolicy resources of the `admission-policy-engine` module. Previously, some settings could be applied incorrectly.
- Revised ModSecurity extension implementation. OWASP Core Rule Set (CRS) functionality is now available in all supported IngressNginxController versions.

Cluster & Infrastructure subsystem

- Lifecycle stages updated for cloud provider integration modules:
  - `cloud-provider-dvp` (DVP), `cloud-provider-vcd` (VMware Cloud Director), `cloud-provider-huaweicloud` (Huawei Cloud), and `cloud-provider-zvirt` (zVirt) modules are now General Availability (previously Experimental).
- Added support for swap memory usage by pods. Instead of immediate OOM-kill, a pod can temporarily use swap (if configured and available on the node) and continue running after peak usage. Requires cgroup v2 and sufficient node resources. For details, see the `kubelet.memorySwap` parameter in the NodeGroup resource documentation.
- In the DVP cloud provider:
  - Added the `networkPolicy` parameter to define network traffic rules for workloads within a Project resource.
  - Added `managed-by`, `cluster-uuid`, and `vm_name` labels to all infrastructure objects.
  - Improved CSI driver diagnostics. Diagnostic messages are now displayed directly in

    PersistentVolumeClaim status.
    - Implemented automatic cleanup of VMBDA objects when a VM is deleted.
    - Fixed an issue where DeckhouseMachine objects could remain in deleting state after VM removal.
    - Fixed an issue where Cloud Controller Manager (CCM) could recreate external load balancers after the associated Kubernetes Service was deleted.

- In Yandex Cloud, fixed a false NAT warning displayed during installation in configurations where NAT is not used.
- In the `node-manager` module:
    - Cluster API (CAPI) has been updated from 1.10.6 to 1.11.3, improving stability and provider compatibility.
    - Added a dedicated controller for NodeGroup hooks (extracted from DKP), reducing load on master nodes and improving responsiveness and UI feedback.
    - Added per-GPU MIG configuration support. You can now define a distinct set of MIG partitions for each physical GPU on a node. Previously, MIG configuration was applied at the node level and identical across all GPUs. This enables more flexible workload resource allocation on a single node.

Network subsystem

- In IngressNginxController, the `acceptClientIPHeadersFrom` parameter has been added for the following inlets: HostPort, HostPortWithProxyProtocol, LoadBalancer, and LoadBalancerWithProxyProtocol. The parameter defines trusted CIDR ranges for `X-Forwarded-*` headers that should not be overwritten during proxying.
- In the `cni-cilium` module, network observability capabilities have been extended:
    - Added extended metrics for monitoring interactions between specific objects (Pods, Services, Namespaces), configurable via the `extendedMetrics` parameter of the HubbleMonitoringConfig resource.
    - Implemented export of traffic flow logs for analyzing network interactions, configurable via the `flowLogs` parameter of the HubbleMonitoringConfig resource.
- Fixed an issue where, after VM migration in DVP, a LoadBalancer IP address could stop being announced when using MetalLB in BGP mode.
- IngressNginxController version support changes:
    - Support for v1.9 has been discontinued.
    - Support for v1.14 has been added.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

Version 1.74

Important

- Support for Kubernetes 1.34 has been added, while support for Kubernetes 1.29 has been discontinued. In future DKP releases, support for Kubernetes 1.30 will be removed. The default Kubernetes version (used when the `kubernetesVersion` parameter is set to `Automatic`) has been changed to 1.32.

Major changes

- Added support for managing experimental and alpha Kubernetes features (feature gates) using the `enabledFeatureGates` parameter of the `control-plane-manager` module. For details on managing these features in DKP, refer to the module documentation.
- The `--terminated-pod-gc-threshold` parameter (the termination threshold for evicted pods) is now calculated automatically based on the number of nodes in the cluster. This provides a balance between cluster stability and the ability to perform diagnostics. Threshold values are available in the `control-plane-manager` module documentation.
- The maximum number of pods (`maxPods`) for a node group is now set automatically based on the `podSubnetNodeCIDRPrefix` parameter from ClusterConfiguration (unless explicitly specified). This ensures stable cluster operation.
- Added the `vipAddress` parameter to the HuaweiCloudInstanceClass resource of the Huawei Cloud provider, allowing you to specify a virtual IP address for all nodes in the instance class.
- The Huawei Cloud provider configuration now allows overriding the primary network for the main network interface (`mainNetwork`) and specifying subnets for additional network interfaces (`additionalNetworks`) on both CloudPermanent and CloudEphemeral nodes.

  Warning. When upgrading to DKP 1.74 on Huawei Cloud, a bulk recreation of CloudEphemeral nodes may occur. Due to a temporary node shortage, some Pods may fail to start.
- The archive with debugging data is now collected using the `d8` tool. If you need to collect diagnostic data, follow the instructions.
- Now when hook execution fails, all metrics collected by that hook are preserved. This prevents loss of debugging information and simplifies the following root-cause analysis.
- Fixed a Deckhouse queue freeze in hybrid clusters that occurred when module configuration parameters

were incomplete.

Security

- Implemented module integrity control based on the EROFS filesystem to improve security and prevent unauthorized modifications. The new system requires kernel support for dm-verity and EROFS.

Network

- Added support for the SCTP (Stream Control Transmission Protocol). Applications using this protocol can now run in a Deckhouse cluster.
- Added a `geoproxy` auxiliary microservice to the `ingress-nginx` module. This service is aimed at improving the stability of the Ingress NGINX Controller when working with GeoIP databases and provides the following features:
  - MaxMind license saving (databases are downloaded from a single point once a day).
  - Persistent data storage (if components are restarted, it doesn't require accessing the MaxMind servers).
  - Lets you specify a custom mirror for downloading databases.
- Added the Prometheus metric `bpf_progs_complexity_max_verified_insts` to assess the number of instructions in eBPF programs loaded into the kernel on cluster nodes. This metric helps evaluate the compatibility of the networking subsystem with the node kernels. The metric requires Linux kernel version 5.16 or newer.
- Added a new metric (`geoip_version`) and dashboards for monitoring GeoIP functionality in the cluster.
- Fixed issues with simultaneous updates of the `node-local-dns` module and the networking subsystem.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

Version 1.73

Important

- This release includes several important security improvements. Multiple known vulnerabilities have been fixed, including one in the user-authn module (CVE-2025-22868) that could potentially allow bypassing authentication checks. It is recommended that you schedule this update. See the Security section for details.
- The `dashboard` module will be removed in future DKP versions. Use the Deckhouse web UI instead (requires the `console` module to be enabled).
- The `runtime-audit-engine` module is now loaded from an external source (the `deckhouse` ModuleSource).
- All DKP components will be restarted during the update.

Major changes

- Added support for changing the registry settings (via the `mode` parameter) in `Unmanaged` mode, similar to `Direct` mode. A new parameter `checkMode` has been added to control registry validation behavior:
  - `Default`: Verifies the availability of all system module images and the deckhouse-controller images.
  - `Relax`: Verifies only the deckhouse-controller images.
- Updated requirements for the `email` field of the User object: the email address must now be in lowercase. Existing users will not be affected.
- Dex updated to **v2.44.0**. It now allows authentication through available identity providers if one of them is down, and supports authentication via identity providers through a proxy.
- The User object status now displays the reason for user lockout (controlled by the `lockout` parameter).
- Added the `additionalDisks` parameter for the Deckhouse Virtualization Platform integration provider, allowing creation and attachment of additional disks to VMs in a NodeGroup (`size` and StorageClass must be specified). This simplifies data distribution across multiple disks.
- Added support for `additionalMetadata` in objects (networks, VMs, disks) for the VMware Cloud Director integration provider. Metadata is merged with existing data, with `additionalMetadata` values taking precedence. Changing `additionalMetadata` triggers recreation of CloudEphemeral node groups that use the affected VCDInstanceClass.
- For the VMware vSphere integration provider, you can now specify an SPBM storage policy ID (via the `storagePolicyID` parameter) and configure automatic creation of a StorageClass for each available SPBM policy. You can now explicitly select a policy for master and worker nodes and use the corresponding storage classes.
- Added alerts to help plan module deprecation or migration:
  - `ModuleIsDeprecated`: Notifies when a module is deprecated and nearing end of support.
  - `D8ModuleOutdatedByMajorVersion`: Notifies when a module is behind by one or more major versions.

- Added `GeoIPDownloadErrorDetected` alert to notify about MaxMind GeoIP database download issues.
- The update notification workflow has changed — a release is applied only after the notification is successfully delivered to the configured webhook. If delivery fails, the update is paused until the webhook is restored.
- Reorganized in-cluster documentation. All module documentation (including connected ones) is now located under the Modules section. Search has been updated.
- For Ingress NGINX Controller v1.10, added the option to enable the profiler (via the `nginxProfilingEnabled` parameter). Enabling the profiler increases resource consumption but may be useful for debugging controller issues.
- Added support for custom HTTP authentication headers (via the `headers` parameter of UpmeterRemoteWrite) when sending SLA monitoring metrics via Prometheus Remote Write protocol.
- Optimized DKP core module loading, reducing startup time in clusters with many modules.
- Audit logs now show which OIDC provider issued the authentication token
- Deckhouse CLI (`d8`) updated to v0.23.0:
  - Added the `d8 status` command, which provides a quick cluster summary (nodes, releases, Deckhouse pods, alerts, registry, Deckhouse settings, CNI, queue state).
  - Added the `d8 k debug` command, which runs the DKP built-in debug container (image can be overridden via `--image`). This simplifies interactive pod debugging.
  - Added the `--watch` flag to the `d8 system queue list` command to track queue state changes.

Security

- Improved baseline container security. Updated security profiles for the following modules to restrict privileges and access rights to the minimum required: `cni-cilium`, `cni-flannel`, `cni-simple-bridge`, `ingress-nginx`, `istio`, `keepalived`, `kube-dns`, `kube-proxy`, `node-local-dns`, `network-gateway`, `network-policy-engine`, `open-vpn`.
- Added the `allowRbacWildcards` flag to the SecurityPolicy, controlling whether wildcards are allowed in Role and RoleBinding definitions (set to `true` by default). Security policies can now also restrict interactive connections to pods (`CONNECT` for `pods/exec` and `pods/attach`) within namespaces.
- Added support for preventing creation of pods with specific tolerations from a list (`policies.disallowedTolerations` parameter in the operational policy). This helps prevent user workloads from running on nodes reserved for special tasks.
- Enhanced security in Ingress NGINX Controller v1.12 (distroless image, vulnerability fixes, and other improvements).
- Fixed known vulnerabilities in the following modules: `operator-trivy`, `registry`, `user-authn`, `cloud-provider-dvp`, `multitenancy-manager`, `admission-policy-engine`, `ingress-nginx`, `alertmanager`, `metallb`, `istio`, `node-local-dns`, `kube-apiserver`.

Network

- For the VMware Cloud Director integration provider, added support for LoadBalancer backed by VMware NSX Advanced Load Balancer (ALB/Avi) when using `NSX-T`. Requires the Load Balancer feature to be enabled on the Edge Gateway. If LoadBalancer is enabled after cluster creation, DKP automatically detects and applies the change within an hour. For open ports, DKP creates corresponding `Pool + Virtual Service` pairs. If there is a firewall, add allow rules for the LoadBalancer's external IP address and relevant ports.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

Version 1.72

Important

- All DKP components will be restarted during the update.
- To use experimental modules in the cluster, you now need to explicitly enable the allowExperimentalModules parameter. By default, experimental modules are disabled. Modules that were enabled before the update will not be automatically disabled. However, if an experimental module enabled prior to the update is manually disabled during the update process, you will need to grant permission to use experimental modules again in order to re-enable it.
- If there are WireGuard interfaces on the cluster nodes, you must update the Linux kernel to version 6.8 or higher.

Major changes

- Added a new registry module and the ability to adjust container registry parameters without restarting all DKP components. Two modes for working with the container registry are now available in DKP: `Unmanaged` (the approach used in previous versions) and `Direct` (a new mode). In `Direct` mode, DKP creates a virtual container registry address in the cluster that all DKP components use. Changing the container registry address (for example, switching to a different registry or changing the DKP edition in the cluster) in this

mode does not trigger a forced restart of all DKP components.

- Added support for recursive DNS servers (configured via the recursiveSettings section of the `cert-manager` module). They are used to verify the existence of a DNS record before starting the ACME DNS-01 domain ownership validation process. This is useful if the same domain is used both publicly and within the cluster, or if the domain has dedicated authoritative DNS servers.
- Introduced separation of modules into critical and functional using the `critical` flag in `module.yaml`. Critical modules are started first. Functional modules are started after the bootstrap process is complete. Their tasks run in parallel and do not block the queue in case of failure. This speeds up cluster installation and improves fault tolerance when starting modules.
- You can now enable logging of all DNS queries (the enableLogs parameter of the `node-local-dns` module).
- In the `cloud-provider-vcd` module, a new WithNAT layout has been added for cluster deployment. It automatically configures NAT and, if necessary, firewall rules for accessing nodes through a bastion host. It also supports both `NSX-T` and `NSX-V`. This makes it possible to deploy a cluster in VMware Cloud Director without pre-configuring the environment (unlike the `Standard` layout).

Security

- Added the fields `user-authn.deckhouse.io/name` and `user-authn.deckhouse.io/preferred_username` to Kubernetes audit log events. These fields display user claims from the OIDC provider, improving authentication monitoring and troubleshooting.
- Kubernetes versions 1.30–1.33 have been updated to the latest patch releases.
- For the AWS provider, added the ability to disable the creation of default security groups (the disableDefaultSecurityGroup parameter). When disabled, security groups must be created manually and explicitly specified in AWSClusterConfiguration, AWSInstanceClass, and NodeGroup. This new feature provides greater control over security settings.
- Added support for password policies for local users (configured in the passwordPolicy section). You can now enforce a minimum password complexity, set password expiration, require password rotation, prevent reuse of old passwords, and lock accounts after a specified number of failed login attempts. These changes allow administrators to centrally enforce password requirements and improve cluster security.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

Version 1.71

Important

- Prometheus has been replaced with Deckhouse Prom++. If you want to keep using Prometheus, disable the `prompp` module manually before upgrading DKP by running the command `d8 system module disable prompp`.
- Support for Kubernetes 1.33 has been added, while support for Kubernetes 1.28 has been discontinued. In future DKP releases, support for Kubernetes 1.29 will be removed. The default Kubernetes version (used when the `kubernetesVersion` parameter is set to `Automatic`) has been changed to 1.31.
- Upgrading the cluster to Kubernetes 1.31 requires a sequential update of all nodes, with each node drained. You can control how node updates requiring workload disruptions are applied using the `disruptions` parameter section.
- The built-in `snapshot-controller` and `static-routing-manager` modules will now be replaced with their external counterparts of the same name, sourced via ModuleSource `deckhouse`.
- The new version of Cilium requires nodes to run Linux kernel version 5.8 or newer. If any node in the cluster has a kernel older than 5.8, the Deckhouse Kubernetes Platform upgrade will be blocked. Cilium Pods will be restarted.
- All DKP components will be restarted during the update.

Major changes

- You can now enforce two-factor authentication for static users. This is configured via the `staticUsers2FA` parameter section of the `user-authn` module.
- Added support for GPUs on nodes. Three GPU resource sharing modes are now available: Exclusive (no sharing), TimeSlicing (time-based sharing), and MIG (a single GPU split into multiple instances). The NodeGroup spec.gpu parameter section is used to configure the GPU resource sharing mode. Using a GPU on a node requires installing the NVIDIA Container Toolkit and the GPU driver.
- When enabling a module (with `d8 system module enable`) or editing a ModuleConfig resource, a warning is now displayed if multiple module sources are found. In such a case, explicitly specify the module source using the source parameter in the module's configuration.
- Improved error handling for module configuration. Module-related errors no longer block DKP operations. Instead, they are now displayed in the status fields of Module and ModuleRelease objects.

- Improved virtualization support:
  - Added a provider for integration with Deckhouse Virtualization Platform (DVP), enabling deployment of DKP clusters on top of DVP.
  - Added support for nested virtualization on nodes in the `cni-cilium` module.
- The `node-manager` module now includes several enhancements for improved node reliability and manageability:
  - You can now prevent a node from restarting if it still hosts critical Pods (labeled with `pod.deckhouse.io/inhibit-node-shutdown`). This can be necessary for workloads with stateful components, such as long-running data migrations.
  - Introduced API version `v1alpha2` for the SSHCredential resource, where the `sudoPasswordEncoded` parameter allows specifying the `sudo` password in Base64 format.
  - The `capiEmergencyBrake` parameter allows you to disable Cluster API (CAPI) in emergency scenarios, preventing potentially destructive changes. Its behavior is similar to the existing `mcmEmergencyBrake` setting.
- Added a pre-installation check to verify connectivity to the DKP container image registry.
- Improved the log file rotation mechanism when using short-term log storage (via the `loki` module). Added the `LokiInsufficientDiskForRetention` alert to warn about insufficient disk space for log retention.
- The documentation now includes a reference for the Deckhouse CLI (`d8` utility) commands and parameters.
- When using CEF encoding for collecting logs from Apache Kafka or socket sources, you can now configure auxiliary CEF fields such as Device Product, Device Vendor, and Device ID.
- The `passwordHash` field in the NodeUser resource is no longer required. This allows you to create users without passwords — for example, in clusters that use external authentication systems (such as PAM or LDAP).
- Added support for CRI Containerd v2 with CgroupsV2. The new version introduces a different configuration format and includes a mechanism to migrate between Containerd v1 and v2. You can change the CRI type used on nodes via the `cri.type` parameter and configure it using `cri.containerdV2`.

Security

- Container image signature verification is now available in DKP SE+. This feature is now supported in DKP SE+ and EE.
- The `log-shipper`, `deckhouse-controller`, and `Istio` (version 1.21) modules have been migrated to distroless builds. This improves security and ensures a more transparent and controlled build process.
- New audit rules have been added to track interactions with containerd. The following are now monitored: access to the `/run/containerd/containerd.sock` socket, modifications to the `/etc/containerd` and `/var/lib/containerd` directories and the `/opt/deckhouse/bin/containerd` file.
- Known vulnerabilities have been fixed in the following modules: `loki`, `extended-monitoring`, `operator-prometheus`, `prometheus`, `prometheus-metrics-adapter`, `user-authn`, and `cloud-provider-zvirt`.

Network

- Added support for Istio version 1.25.2, which uses the Sail operator instead of the deprecated Istio Operator. Also added support for Kiali version 2.7, without Ambient Mesh support. Istio version 1.19 is now considered deprecated.
- Added support for encrypting traffic between nodes and Pods using the WireGuard protocol (via the `encryption.mode` parameter).
- Fixed the logic for determining service readiness in the ServiceWithHealthcheck resource. Previously, Pods without an IP address (for example, in `Pending` state) could be mistakenly included in the load balancing list.
- Added support for the least-conn load balancing algorithm. This algorithm directs traffic to the service backend with the fewest active connections, improving performance for connection-heavy applications (such as WebSocket services). To use this algorithm, enable the `extraLoadBalancerAlgorithmsEnabled` parameter in the `cni-cilium` module settings and use the `service.cilium.io/lb-algorithm` annotation on the service and set it to a supported value: random, maglev, or least-conn.
- Fixed an issue in Cilium 1.17 `cilium-operator` where IP addresses were not reused after a `CiliumEndpoint` was deleted. The issue was caused by improper cleanup of priority filters, which could lead to IP pool exhaustion in large clusters.
- Refined the list of ports used for networking:
  - Added and updated:
    - `4287/UDP`: WireGuard port used for CNI Cilium traffic encryption.
    - `4295-4297/UDP`: Used by the `cni-cilium` module for VXLAN encapsulation of inter-pod traffic in multiple nested virtualization — when DKP with the `virtualization` module enabled is deployed inside virtual machines that are also created in DKP with the `virtualization` module enabled.
    - `4298/UDP`: Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods if the cluster was deployed on DKP version starting from **1.71** (for clusters deployed on DKP versions up to **1.71**, see the note for ports `4299/UDP`, `8469/UDP`, and `8472/UDP`).

        - `4299/UDP`: Port **for clusters deployed on DKP versions 1.64–1.70.** Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods. Updating DKP to newer versions will not change the port used unless the `virtualization` module is enabled.
        - `8469/UDP`: Port **for clusters deployed on DKP version 1.63 and below with the `virtualization` module enabled prior to DKP version 1.63.** Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods. Updating DKP to newer versions will not change the occupied port
        - `8472/UDP`: Port **for clusters deployed on DKP version 1.63 and below.** Used by the `cni-cilium` module for VXLAN encapsulation of traffic between pods. Updating DKP to newer versions will not change the occupied port if the `virtualization` module is not enabled. **Note** that in such clusters, enabling the `virtualization` module on DKP before version 1.70 changes the port:
            - Enabling the `virtualization` module on DKP version 1.63 and below will change it to `8469/UDP` and will not change with subsequent DKP updates
            - Enabling the `virtualization` module on DKP starting from version 1.64 will change it to `4298/UDP` and will not change with subsequent DKP updates
    - Removed:
        - `49152`, `49153/TCP`: Previously used for live migration of virtual machines (in the `virtualization` module). Migration now occurs over the Pod network.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

Version 1.70

Important

- The `ceph-csi` module has been removed. Use the `csi-ceph` module instead. Deckhouse will not be updated as long as `ceph-csi` is enabled in the cluster. For `csi-ceph` migration instructions, refer to the module documentation.
- Version 1.12 of the Ingress NGINX Controller has been added. The default controller version has been changed to 1.10. All Ingress controllers that do not have an explicitly specified version (via the `controllerVersion` parameter in the IngressNginxController resource or the `defaultControllerVersion` parameter in the `ingress-nginx` module) will be restarted.
- The `falco events` metric (from the `runtime-audit-engine` module) has been removed. The `falco events` metric was considered deprecated since DKP 1.68. Use the `falcosecurity_falcosidekick_falco_events_total` metric instead. Dashboards and alerts based on the `falco_events` metric may stop working.
- All DKP components will be restarted during the update.

Major changes

- In the `Auto` update mode, patch version updates (for example, from `v1.70.1` to `v1.70.2`) are now applied taking into account the update windows, if they are set. Previously, in this update mode, only minor version updates (for example, from `v1.69.x` to `v1.70.x`) were applied with consideration to update windows, while patch version updates were applied as they appeared on a release channel.
- A node can now be rebooted if the corresponding Node object has the `update.node.deckhouse.io/reboot` annotation set.
- When cleaning up a static node, any local users created by Deckhouse Kubernetes Platform are now also removed.
- Added synchronization monitoring for Istio in multi-cluster configurations. A new alert `D8IstioRemoteClusterNotSynced` has been introduced and triggers in the following cases:
    - The remote cluster is offline.
    - The remote API endpoint is not reachable.
    - The remote `ServiceAccount` token is invalid or expired.
    - There is a TLS or certificate issue between the clusters.
- The `deckhouse-controller collect-debug-info` command now also collects debug information for `Istio`, including:
    - Resources in the `d8-istio` namespace.
    - CRDs from the `istio.io` and `gateway.networking.k8s.io` groups.
    - `Istio` logs.
    - `Sidecar` logs of a single randomly selected user application.
- A new monitoring dashboard has been added to display OpenVPN certificate status. Upon expiration, server certificates will now be reissued, and client certificates will be removed. The following alerts have been added:
    - `OpenVPNClientCertificateExpired`: Warns about expired client certificates.
    - `OpenVPNServerCACertificateExpired`: Warns about an expired OpenVPN CA certificate.
    - `OpenVPNServerCACertificateExpiringSoon` and `OpenVPNServerCACertificateExpiringInAWeek`: Warn when an OpenVPN CA certificate is expiring in less than 30 or 7 days, respectively.
    - `OpenVPNServerCertificateExpired`: Warns about an expired OpenVPN server certificate.
    - `OpenVPNServerCertificateExpiringSoon` and `OpenVPNServerCertificateExpiringInAWeek`: Warn when an OpenVPN server certificate is expiring in less than 30 or 7 days, respectively.
- Monitoring dashboards have been renamed and updated:
    - “L2LoadBalancer” renamed to “MetalLB L2”; pool and column filtering added.
    - “Metallb” renamed to “MetalLB BGP”; pool and column filtering added. The ARP request panel has been removed.
    - “L2LoadBalancer / Pools” renamed to “MetalLB / Pools”.

- The `upmeter` module's PVC size has been increased to accommodate data retention for 13 months. In some cases, the previous PVC size was insufficient.
- The ModuleSource resource status now includes information about module versions in the source.
- The Module resource status now includes information about the module's lifecycle stage. A module can move through the following stages in its lifecycle: Experimental, Preview, General Availability, and Deprecated. For details on module lifecycle stages and how to evaluate its stability, refer to the corresponding section in the documentation.
- It is now possible to use stronger or more modern encryption algorithms (such as `RSA-3072`, `RSA-4096`, or `ECDSA-P256`) for control plane cluster certificates instead of the default `RSA-2048`. You can use the `encryptionAlgorithm` parameter in the ClusterConfiguration resource to configure this.
- The `descheduler` module can now be configured to evict pods that are using local storage. Use the `evictLocalStoragePods` parameter in the module configuration to adjust this.
- You can now adjust the logging level of the Ingress controller using the `controllerLogLevel` parameter in the IngressNginxController resource. The default log level is `Info`. Controlling the logging level can help prevent log collector overload during Ingress controller restarts.

Security

- The severity level of alerts indicating security policy violations has been raised from 7 to 3.
- The configuration for `Yandex Cloud`, `Zvirt`, and `Basis Dynamix` providers now uses `OpenTofu` instead of `Terraform`. This enables easier provider updates, such as applying fixes for known vulnerabilities (CVEs).
- CVE vulnerabilities have been fixed in the following modules: `chrony`, `descheduler`, `dhctl`, `node-manager`, `registry-packages-proxy`, `falco`, `cni-cilium`, and `vertical-pod-autoscaler`.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

Version 1.69

Important

- Support for Kubernetes 1.32 has been added, while support for Kubernetes 1.27 has been discontinued. The default Kubernetes version has been changed to 1.30. In future DKP releases, support for Kubernetes 1.28 will be removed.
- All DKP components will be restarted during the update.

Major changes

- The `ceph-csi` module is now deprecated. Plan to migrate to the `csi-ceph` module instead. For details, refer to the Ceph documentation.
- You can now grant access to Deckhouse web interfaces using user names via the `auth.allowedUserEmails` field. Access restriction is configured together with the `auth.allowedUserGroups` parameter in configuration of the following modules with web interfaces: `cilium-hubble`, `dashboard`, `deckhouse-tools`, `documentation`, `istio`, `openvpn`, `prometheus`, and `upmeter` (example for `prometheus`).
- A new dashboard **Cilium Nodes Connectivity Status & Latency** has been added to Grafana in the `cni-cilium` module. It helps monitor node network connectivity issues. The dashboard displays a connectivity matrix similar to the `cilium-health status` command, using metrics that are already available in Prometheus.
- A new `D8KubernetesStaleTokensDetected` alert has been added in the `control-plane-manager` module that is triggered when stale service account tokens are detected in the cluster.
- You can now create a Project from an existing namespace and adopt existing objects into it. To do this, annotate the namespace and its resources with `projects.deckhouse.io/adopt`. This lets you switch to using Projects without recreating cluster resources.
- A `Terminating` status has been added to ModuleSource and ModuleRelease resources. The new status will be displayed when an attempt to delete one of them fails.
- The installer container now automatically configures cluster access after a successful bootstrap. A `kubeconfig` file is generated in `~/.kube/config`, and a local TCP proxy is set up through an SSH tunnel. This allows you to use kubectl locally right away without manually connecting to the control-plane node via SSH.
- Changes to Kubernetes resources in multi-cluster and federation setups are now tracked directly via Kubernetes API. This enables faster synchronization between clusters and eliminates the use of outdated certificates. In addition, mounting of ConfigMap and Secret resources into Pods has been removed to eliminate family system compromise risks.
- A new dynamicforward plugin has been added to CoreDNS, improving DNS query processing in the cluster. It integrates with `node-local-dns`, continuously monitors `kube-dns` endpoints, and automatically updates

the list of DNS forwarders. If the control-plane node is unavailable, DNS queries are still forwarded to available endpoints, improving cluster stability.

- A new log rotation approach has been introduced in the `loki` module. Now, old logs are automatically removed when disk usage exceeds a threshold: either 95% of PVC size or PVC size minus the size required to store two minutes of log data at the configured ingestion rate (`ingestionRateMB`). The `retentionPeriodHours` parameter no longer controls the data retention and is used for monitoring alerts only. If `loki` begins removing old logs before the set period is reached, a `LokiRetentionPerionViolation` alert will be triggered, informing the user that they must reduce the value of `retentionPeriodHours` or increase the PVC size.
- A new `nodeDrainTimeoutSecond` parameter lets you set the maximum timeout when attempting to drain a node (in seconds) for each NodeGroup resource. Previously, you could only use the default value (10 minutes) or reduce it to 5 minutes using the `quickShutdown` parameter, which is now deprecated.
- The `openvpn` module now includes a `defaultClientCertExpirationDays` parameter, allowing you to define the lifetime of client certificates.

Security

- Known vulnerabilities have been addressed in the following modules: `ingress-nginx`, `istio`, `prometheus`, and `local-path-provisioner`.

The complete list of changes, including the updated components, is available in the changelog on GitHub.

Version 1.68

Important

- After the update, the UID will change for all Grafana data sources created using the GrafanaAdditionalDatasource resource. If a data source was referenced by UID, that reference will no longer be valid.

Major changes

- A new parameter, `iamNodeRole`, has been introduced for the AWS provider. It lets you specify the name of the IAM role to bind to all AWS instances of cluster nodes. This can come in handy if you need to grant additional permissions (for example, access to ECR, etc.).
- Creating nodes of the CloudPermanent type now takes less time. Now, CloudPermanent nodes are created in parallel. Previously, they were created in parallel only within a single group.
- Monitoring changes:
  - Support for monitoring certificates in secrets of the `Opaque` type has been added.
  - Support for monitoring images in Amazon ECR has been added.
  - A bug that could cause partial loss of metrics when Prometheus instances were restarted has been fixed.
- When using a multi-cluster Istio configuration or federation, you can now explicitly specify the list of addresses used for inter-cluster requests. Previously, these addresses were determined automatically; however, in some configurations, they could not be resolved.
- The DexAuthenticator resource now has a `highAvailability` parameter that controls high availability mode. In high availability mode, multiple replicas of the authenticator are launched. Previously, high availability mode of all authenticators was determined by a global parameter or by the `user-authn` module. All authenticators deployed by DKP now inherit the high availability mode of the corresponding module.
- Node labels can now be added, removed, or modified using files stored on the node in the `/var/lib/node labels` directory and its subdirectories. The full set of applied labels is stored in the `node.deckhouse.io/last-applied-local-labels` annotation.
- Support for the Huawei Cloud provider has been added.
- The new `keepDeletedFilesOpenedFor` parameter in the `log-shipper` module allows you to configure the period to keep the deleted log files open. This way, you can continue reading logs from deleted pods for some time if log storage is temporarily unavailable.
- TLS encryption for log collectors (Elasticsearch, Vector, Loki, Splunk, Logstash, Socket, Kafka) can now be configured using secrets, rather than by storing certificates in the ClusterLogDestination resources. The secret must reside in the `d8-log-shipper` namespace and have the `log-shipper.deckhouse.io/watch-secret: true` label.
- In the project status under the `resources` section, you can now see which project resources have been installed. Those resources are marked with `installed: true`.
- A new parameter, `--tf-resource-management-timeout`, has been added to the installer. It controls the resource creation timeout in cloud environments. By default, the timeout is set to 10 minutes. This parameter applies only to the following clouds: AWS, Azure, GCP, OpenStack.

Security

Known vulnerabilities have been addressed in the following modules:

- admission-policy-engine
- chrony
- cloud-provider-azure
- cloud-provider-gcp
- cloud-provider-openstack
- cloud-provider-yandex
- cloud-provider-zvirt
- cni-cilium
- control-plane-manager
- extended-monitoring
- descheduler
- documentation
- ingress-nginx
- istio
- loki
- metallb
- monitoring-kubernetes
- monitoring-ping
- node-manager
- operator-trivy
- pod-reloader
- prometheus
- prometheus-metrics-adapter
- registrypackages
- runtime-audit-engine
- terraform-manager
- user-authn
- vertical-pod-autoscaler
- static-routing-manager

Mandatory component restart

The following components will be restarted after updating DKP to 1.68:

- Kubernetes Control Plane
- Ingress controller
- Prometheus, Grafana
- admission-policy-engine
- chrony
- cloud-provider-azure
- cloud-provider-gcp
- cloud-provider-openstack
- cloud-provider-yandex
- cloud-provider-zvirt
- cni-cilium
- control-plane-manager
- descheduler
- documentation
- extended-monitoring
- ingress-nginx
- istio
- kube-state-metrics
- log-shipper
- loki
- metallb
- monitoring-kubernetes
- monitoring-ping
- node-manager
- openvpn
- operator-trivy
- prometheus
- prometheus-metrics-adapter
- pod-reloader
- registrypackages
- runtime-audit-engine
- service-with-healthchecks
- static-routing-manager
- terraform-manager
- user-authn
- vertical-pod-autoscaler

The complete list of changes, including the updated components, is available in the changelog on GitHub.

Embedded Deckhouse Kubernetes Platform modules

The admission-policy-engine module

> **i** Available with limitations in: **CE, BE, SE, SE+**
>
> Available without limitations in: **EE**

> **i** The module lifecycle stage: General Availability

The `admission-policy-engine` module implements support for admission security policies in a Kubernetes cluster.

Admission policies are rules applied to objects (e.g., Pod and Service) at the time of their creation or modification in the cluster (but not during their operation), based on the information provided in their manifest. These policies are aimed at formalizing parameters that are allowed or prohibited in object manifests.

Policies are divided into three categories:

- Pod Security Standards: Policies that comply with the relevant Pod Security Standards.
- Operational policies: Policies for creating additional requirements for objects by validating the values of parameters that are **not directly related** to security (for example, a list of allowed prefixes for container images, an image download policy, a list of required container images, etc.).
- Security policies: Policies for creating additional requirements on objects by validating the values of security-related parameters (e.g., container access to the host's IPC or PID namespaces, privilege lists for containers, etc.).

> **i** These policies complement each other. If multiple policies are applied to a single namespace, objects are validated against each of them. If even one policy is violated, the object will not be created.

How validation failure messages are displayed

Depending on how pods are created, there are differences in how the API generates messages regarding validation failures (violations of established policies):

- If a pod is created directly, the validation error is returned in the API response indicating a validation failure (policy violation).
- If pods are created via Deployment, the required number of ReplicaSets is created, which in turn attempt to create the pods. In this case, the validation error is not returned in the API response but is displayed in the namespace events or the corresponding ReplicaSet events.

Pod validation when policies are modified or new ones are added

For all three policy categories (Pod Security Standards, operational, and security policies), there is no provision for automatically recreating existing pods when changing existing policies or adding new ones. Pods that existed prior to changes being made to the policy in use or prior to a new policy being added will continue to run until they are restarted. Upon restart, they will be validated against the new rules.

The `admission-policy-engine` module provides alerts (`kind: ClusterObservabilityAlert`) for such cases, notifying you of pods in the namespace that violate policies after an existing policy is modified or a new one is added.

To get a list of alerts, use the command:

```
d8 k get clusterobservabilityalerts
```

Output example:

```
NAME                                                SEVERITY   STATUS   DURATION   SUMMARY
AGE
SecurityPolicyViolation-f3a77d1dd2175402-1777370195   1          Firing   5h
Alerting PrometheusUnavailable   5h1m
OperationPolicyViolation-9b21d0c871796913-1777370435  1          Firing   6h
Alerting PrometheusUnavailable   6h1m
```

To view information about a specific alert, use the following command:

```
d8 k get clusterobservabilityalert OperationPolicyViolation-9b21d0c871796913-1777370435 -oyaml
```

Example of an alert for a violation of the Pod Security Standards policy...

```
kind: ClusterObservabilityAlert
apiVersion: alerts.observability.deckhouse.io/v1alpha1
metadata:
  name: PodSecurityStandardsViolation-91e71759e048a397-1777369535
  resourceVersion: "7454828154578800069"
  creationTimestamp: 2026-04-28T09:45:35Z
  labels:
    d8_component: gatekeeper
    d8_module: admission-policy-engine
    prometheus: deckhouse
alert:
  labels:
    alertname: PodSecurityStandardsViolation
    d8_component: gatekeeper
    d8_module: admission-policy-engine
    prometheus: deckhouse
    severity_level: "3"
  annotations:
    description: |-
      You have configured [Pod Security Standards]
(https://kubernetes.io/docs/concepts/security/pod-security-standards/), and one or more
running pods are violating these standards.

      To identify violating pods:

      - Run the following Prometheus query:

        ```prometheus
        count by (violating_namespace, violating_name, violation_msg) (
          d8_gatekeeper_exporter_constraint_violations{
            violation_enforcement="deny",
            violating_namespace=~".*",
            violating_kind="Pod",
            source_type="PSS"
          }
        )
        ```

      - Alternatively, check the admission-policy-engine Grafana dashboard.
    plk_markup_format: markdown
    plk_protocol_version: "1"
    summary: At least one pod violates the configured cluster pod security standards.
  expr:
(count(d8_gatekeeper_exporter_constraint_violations{source_type="PSS",violating_kind="Pod",v
iolating_namespace=~".*",violation_enforcement="deny"}))
    > 0
  created_by: observability
  rule_group_name: admission-policy-engine-audit-0
status:
  alertStatus: Firing
  silencedBy: []
  startsAt: 2026-04-28T09:45:35Z
  resolvedAt: null
  duration: 20h40m1.015261771s
```

Example of an alert for a violation of an operational policy...

```
kind: ClusterObservabilityAlert
apiVersion: alerts.observability.deckhouse.io/v1alpha1
metadata:
  name: OperationPolicyViolation-9b21d0c871796913-1777370435
  resourceVersion: "7454831929456594373"
  creationTimestamp: 2026-04-28T10:00:35Z
  labels:
    d8_component: gatekeeper
    d8_module: admission-policy-engine
    prometheus: deckhouse
alert:
  labels:
    alertname: OperationPolicyViolation
    d8_component: gatekeeper
    d8_module: admission-policy-engine
    prometheus: deckhouse
    severity_level: "3"
  annotations:
    description: >-
      You have configured operation policies for the cluster, and one or more
      existing objects are violating these policies.


      To identify violating objects:


      - Run the following Prometheus query:

        ```prometheus
        count by (violating_namespace, violating_kind, violating_name, violation_msg) (
          d8_gatekeeper_exporter_constraint_violations{
            violation_enforcement="deny",
            source_type="OperationPolicy"
          }
        )
        ```

      - Alternatively, check the admission-policy-engine Grafana dashboard.
    plk_markup_format: markdown
    plk_protocol_version: "1"
    summary: At least one object violates the configured cluster operation policies.
  expr:
(count(d8_gatekeeper_exporter_constraint_violations{source_type="OperationPolicy",violation_
enforcement="deny"}))
    > 0
  created_by: observability
  rule_group_name: admission-policy-engine-audit-0
status:
  alertStatus: Firing
  silencedBy: []
  startsAt: 2026-04-28T10:00:35Z
  resolvedAt: null
  duration: 20h23m41.023025059s
```

Example of an alert for a security policy violation...

```
kind: ClusterObservabilityAlert
apiVersion: alerts.observability.deckhouse.io/v1alpha1
metadata:
  name: SecurityPolicyViolation-f3a77d1dd2175402-1777370195
  resourceVersion: "7454830922622307781"
  creationTimestamp: 2026-04-28T09:56:35Z
  labels:
    d8_component: gatekeeper
    d8_module: admission-policy-engine
    prometheus: deckhouse
alert:
  labels:
    alertname: SecurityPolicyViolation
    d8_component: gatekeeper
    d8_module: admission-policy-engine
    prometheus: deckhouse
    severity_level: "3"
  annotations:
    description: >-
      You have configured security policies for the cluster, and one or more
      existing objects are violating these policies.


      To identify violating objects:


      - Run the following Prometheus query:

        ```prometheus
        count by (violating_namespace, violating_kind, violating_name, violation_msg) (
          d8_gatekeeper_exporter_constraint_violations{
            violation_enforcement="deny",
            source_type="SecurityPolicy"
          }
        )
        ```

      - Alternatively, check the admission-policy-engine Grafana dashboard.
    plk_markup_format: markdown
    plk_protocol_version: "1"
    summary: At least one object violates the configured cluster security policies.
  expr:
(count(d8_gatekeeper_exporter_constraint_violations{source_type="SecurityPolicy",violation_e
nforcement="deny"}))
    > 0
  created_by: observability
  rule_group_name: admission-policy-engine-audit-0
status:
  alertStatus: Firing
  silencedBy: []
  startsAt: 2026-04-28T09:56:35Z
  resolvedAt: null
  duration: 20h29m21.015479019s
```

Pod Security Standards

Pod Security Standards (`PSS`) is an official Kubernetes standard that defines three security levels for pods, limiting their privileges. Restrictions are enforced by prohibiting the setting of certain parameters in the pod manifest.

A layered structure is used — each higher level of protection uses all the rules of the previous level and adds its own.

The following protection levels are regulated:

- `Privileged`: An unrestricted policy with the widest possible level of permissions (no restrictions).
- `Baseline`: A minimally restrictive policy that prevents the most known and popular ways of privilege escalation. Allows using the standard (minimally specified) pod configuration.
- `Restricted`: A policy with significant restrictions. Imposes the strictest requirements on pods.

> **i** In the Deckhouse Kubernetes Platform, these policies are implemented using Gatekeeper and enforced by the admission controllers of the `admission-policy-engine` module, rather than the Kubernetes Pod Security Admission controller. Only the policy descriptions are taken from Kubernetes.

You can read more about each set of policies and their restrictions in the Kubernetes documentation.

Configuring PSS policies for namespaces is done by setting a special label `security.deckhouse.io/pod-policy=<POLICY_NAME>` on the corresponding namespace. The default policy can be overridden globally (in the module settings).

> **i** The module does not apply policies to system namespaces.

> **i** When the `multitenancy-manager` module is enabled, it creates its own OperationPolicy objects (for example, in the `default` namespace). These are not affected by the `podSecurityStandards` settings.

Example of setting the `Restricted` policy for all pods in the `my-namespace` namespace:

```
d8 k label ns my-namespace security.deckhouse.io/pod-policy=restricted
```

Additionally, it is possible to configure the policy enforcement mode. The following modes are supported:

- `deny`: Prohibit starting pods that do not satisfy the policy.
- `warn`: Start pods that do not satisfy the policy, but issue a warning.
- `dryrun`: Start pods that do not satisfy the policy, do not issue a warning to the user, but record violations in security reports.

Configuring the policy enforcement mode is done by setting the label `security.deckhouse.io/pod-policy-action=<POLICY ACTION>` on the corresponding namespace. To set the policy enforcement mode globally, use the `enforcementaction` parameter.

Example of setting the “warn” mode for PSS policies for all pods in the `my-namespace` namespace:

```
d8 k label ns my-namespace security.deckhouse.io/pod-policy-action=warn
```

Operational policies

Operational policies are rules aimed at achieving application security best practices, but not directly related to the validation of classic security-related parameters (for example, a list of allowed prefixes for container images, an image download policy, a list of required container images, etc.).

Operational policies are described using the `OperationPolicy` custom resource. In this resource, each parameter is responsible for a separate check applied to resources. Using the OperationPolicy custom resource allows you to define additional requirements for the resources being created (high-level declarative operational policies) without explicitly interacting with Gatekeeper.

We recommend setting the following minimum set of operational policies:

```
apiVersion: deckhouse.io/v1alpha1
kind: OperationPolicy
metadata:
  name: common
spec:
  enforcementAction: Deny
  policies:
    allowedRepos:
      - myrepo.example.com
      - registry.deckhouse.ru
    requiredResources:
      limits:
        - memory
      requests:
        - cpu
        - memory
    disallowedImageTags:
      - latest
    requiredProbes:
      - livenessProbe
      - readinessProbe
    maxRevisionHistoryLimit: 3
    imagePullPolicy: Always
    priorityClassNames:
    - production-high
    - production-low
    checkHostNetworkDNSPolicy: true
    checkContainerDuplicates: true
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          custom-operation-policy/enabled: "true"
```

Policy application is implemented through settings located in the `spec.match` parameter.

When specifying:

```
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          custom-operation-policy/enabled: "true"
```

To apply the above policy, it is sufficient to add the label `custom-operation-policy/enabled: "true"` to the desired namespace. Unlike `PSS`, the label name can be anything. Only a match between the label in the policy selector and the corresponding namespace is required.

You can read more detailed information about using selectors in the selector setup description.

It is also possible to specify the action to be applied for the policy. The `spec.enforcementAction` parameter is used for this. The following modes are supported:

- `Deny`: Prohibit starting pods that do not satisfy the policy.
- `Warn`: Start pods that do not satisfy the policy, but issue a warning.
- `Dryrun`: Start pods that do not satisfy the policy, do not issue a warning to the user, but record violations in security reports.

Based on this example, you can create your own policy with the necessary settings.

Security policies

Security policies are rules aimed at achieving application security best practices by validating the values of security-related parameters.

Security policies are described using the `SecurityPolicy` custom resource. In this resource, each parameter is responsible for a separate check applied to resources. Using this resource, it is possible to construct a security policy similar to a PSS policy of any level. Using the custom SecurityPolicy resource allows you to define additional requirements for the resources being created (high-level declarative security policies) without explicitly interacting with Gatekeeper.

Example of a security policy:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: mypolicy
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: true
    allowHostNetwork: true
    allowHostPID: false
    allowPrivileged: false
    allowPrivilegeEscalation: false
    allowedFlexVolumes:
    - driver: vmware
    allowedHostPorts:
    - max: 4000
      min: 2000
    allowedProcMount: Unmasked
    allowedAppArmor:
    - unconfined
    allowedUnsafeSysctls:
    - kernel.*
    allowedVolumes:
    - hostPath
    - projected
    fsGroup:
      ranges:
      - max: 200
        min: 100
      rule: MustRunAs
    readOnlyRootFilesystem: true
    requiredDropCapabilities:
    - ALL
    runAsGroup:
      ranges:
      - max: 500
        min: 300
      rule: RunAsAny
    runAsUser:
      ranges:
      - max: 200
        min: 100
      rule: MustRunAs
    seccompProfiles:
      allowedLocalhostFiles:
      - my_profile.json
      allowedProfiles:
      - Localhost
    supplementalGroups:
      ranges:
      - max: 133
        min: 129
      rule: MustRunAs
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          security-policy: mypolicy
```

⚠ The `allowPrivilegeEscalation` and `allowPrivileged` parameters default to `false` — even if not explicitly specified. This means that containers will not be able to run in privileged mode or escalate privileges. To allow such behavior, set the parameter to `true`.

Policy application is implemented through settings located in the `spec.match` parameter.

When specifying:

```
match:
  namespaceSelector:
    labelSelector:
      matchLabels:
        security-policy: mypolicy
```

To apply the above policy, it is sufficient to add the label `security-policy: mypolicy` to the desired namespace. Unlike `PSS`, the label name can be anything. Only a match between the label in the policy selector and the corresponding namespace is required.

You can read more detailed information about using selectors in the selector setup description.

It is also possible to specify the action to be applied for the policy. The `spec.enforcementAction` parameter is used for this. The following modes are supported:

- `Deny`: Prohibit starting pods that do not satisfy the policy.
- `Warn`: Start pods that do not satisfy the policy, but issue a warning.
- `Dryrun`: Start pods that do not satisfy the policy, do not issue a warning to the user, but record violations in security reports.

Modifying Kubernetes resources

The module allows you to use the Gatekeeper Custom Resources to modify objects in the cluster, such as:

- AssignMetadata — defines changes to the `metadata` section of a resource.
- Assign — any change outside the `metadata` section.
- ModifySet — adds or removes entries from a list, such as the arguments to a container.
- AssignImage — to change the `image` parameter of the resource.

You can read more about the available options in the gatekeeper documentation.

Third-party components

List of third-party software used in the `admission-policy-engine` module:

- Gatekeeper 3.22.0

  License: Apache License 2.0

  Policy-based control for cloud native environments

Configuration

> **i** Available with limitations in: **CE, BE, SE, SE+**
>
> Available without limitations in: **EE**

> **i** The module lifecycle stage: General Availability

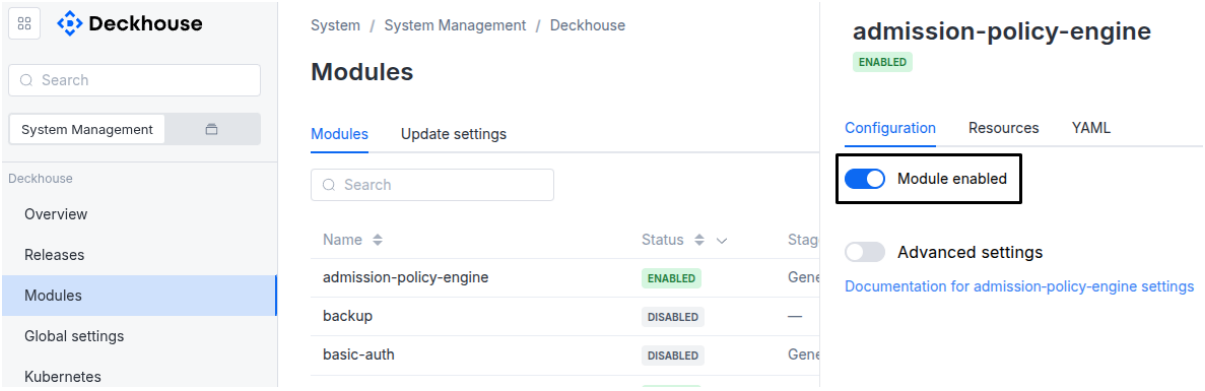
The module has 4 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `admission-policy-engine` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable admission-policy-engine
  ```

- **Using ModuleConfig `admission-policy-engine`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `admission-policy-engine` (create it if necessary);

  Example of a manifest to enable module `admission-policy-engine`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: admission-policy-engine
  spec:
    enabled: true
  ```

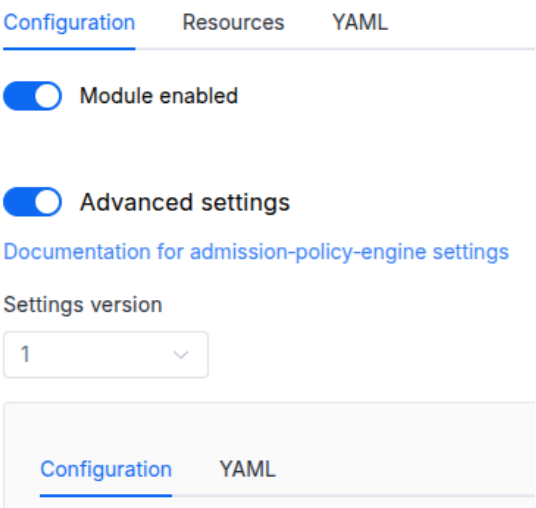
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `admission-policy-engine` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `admission-policy-engine` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `admission-policy-engine`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

Edit the existing ModuleConfig `admission-policy-engine` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc admission-policy-engine
```

Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `admission-policy-engine` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `admission-policy-engine`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: admission-policy-engine
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.denyVulnerableImages**
    object

    Available in editions: EE

    Trivy provider settings.

    The Trivy provider prevents creation of the Pod, Deployment, StatefulSet, and DaemonSet resources with images containing vulnerabilities in namespaces labeled with `security.deckhouse.io/trivy-provider: ""`.

    Deprecated. These settings have been moved to the `operator-trivy` module. Please migrate your configuration.

    - **settings.denyVulnerableImages.allowedSeverityLevels**
      array of strings

      List of allowed vulnerability severity levels.

      Images containing only vulnerabilities of the specified severity levels will not be blocked.

      - **Element of the array**
        string

        Allowed values: `UNKNOWN`, `LOW`, `MEDIUM`, `HIGH`, `CRITICAL`

    - **settings.denyVulnerableImages.enabled**
      boolean

      Enable the Trivy provider.

      Default: `false`

    - **settings.denyVulnerableImages.registrySecrets**
      array of objects

      List of additional Secrets for accessing private registries.

      By default, the `deckhouse-registry` Secret is used to download images for scanning.

      Default: `[]`

      - **settings.denyVulnerableImages.registrySecrets.name**
        string

        Required value

- **settings.denyVulnerableImages.registrySecrets.namespace**
  string

  Required value

- **settings.denyVulnerableImages.storageClass**
  string

  Name of the StorageClass used by the Trivy provider.

  If the value is not specified, the StorageClass set in the global `settings.modules.storageClass` parameter is used instead.

  The global `settings.modules.storageClass` parameter is only applied when the module is enabled. Changing the global parameter while the module is enabled will not trigger disk re-provisioning.

  > **Warning.** If you specify a value different from the current StorageClass used in the existing PersistentVolumeClaim, the disk will be re-provisioned, and all data will be deleted.

  If you set it to `false`, `emptyDir` will be forcibly used instead of a PVC.

  Examples:

  ```
  storageClass: ceph-ssd
  ```

  ```
  storageClass: 'false'
  ```

- **settings.podSecurityStandards**
  object

  Pod Security Standards policy settings (PSS).

  - **settings.podSecurityStandards.defaultPolicy**
    string

    Sets the default Pod Security Standards policy for all **non-system** namespaces:

    - `Privileged` — an unrestricted policy. Privilege escalation is possible with this policy;
    - `Baseline` — a policy with minimum restrictions that prevents privilege escalation;
    - `Restricted` — a policy with maximum restrictions that conforms to current best practices for securely running applications in a cluster.

    By default:

    - `Baseline` — if a Deckhouse version starting with v1.55 is being installed;
    - `Privileged` — if a Deckhouse version lower than v1.55 is being installed (upgrading Deckhouse in a cluster to v1.55+ does not automatically result in a default policy change).

    Allowed values: `Privileged`, `Baseline`, `Restricted`

  - **settings.podSecurityStandards.enforcementAction**
    string

    The enforcement action to control what to do with the result of the constraint.

    - Deny — Deny action.
    - Dryrun — No action. It is used when debugging. Information about the event can be viewed in Grafana or in the console via `d8 k`.
    - Warn — Same as `Dryrun`. In addition to the event information, it provides some info on why that constraint would have been denied if you had set `Deny` instead of `Warn`.

    Default: `Deny`

    Allowed values: `Warn`, `Deny`, `Dryrun`

  - **settings.podSecurityStandards.policies**
    object

    Sets additional policy parameters.

    - **settings.podSecurityStandards.policies.hostPorts**
      object

      HostPort constraint settings.

      - **settings.podSecurityStandards.policies.hostPorts.knownRanges**
        array of objects

        Set the range of known ports which will be allowed in a hostPort binding.

- **settings.podSecurityStandards.policies.hostPorts.knownRanges.max**
  integer
- **settings.podSecurityStandards.policies.hostPorts.knownRanges.min**
  integer

Custom resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

OperationPolicy

Scope: Cluster
Version: v1alpha1

Describes an operation policy for a cluster.

Each CustomResource `OperationPolicy` describes rules for objects in a cluster.

- **spec**
  object

  Required value

  - **spec.enforcementAction**
    string

    The enforcement action to control what to do with the result of the constraint.

    - `Deny` — Deny action.
    - `Dryrun` — No action. It is used when debugging. Information about the event can be viewed in Grafana in dashboard `Security/Admission policy engine`.
    - `Warn` — Same as `Dryrun`. In addition to the event information, it provides some info on why that constraint would have been denied if you had set `Deny` instead of `Warn`.

    Default: `Deny`

    Allowed values: `Warn`, `Deny`, `Dryrun`

  - **spec.match**
    object

    Required value

    - **spec.match.labelSelector**
      object

      Specifies the label selector to filter Pods with.

      You can get more into here.

      - **spec.match.labelSelector.matchExpressions**
        array of objects

        List of label expressions for Pods.

        Example:

        ```
        matchExpressions:
        - key: tier
          operator: In
          values:
          - production
          - staging
        ```

        - **spec.match.labelSelector.matchExpressions.key**
          string

          Required value

        - **spec.match.labelSelector.matchExpressions.operator**
          string

          Required value

Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.match.labelSelector.matchExpressions.values**
  array of strings
- **spec.match.labelSelector.matchLabels**
  object

  List of labels which Pod should have.

  Example:

  ```
  matchLabels:
    foo: bar
    baz: who
  ```

- **spec.match.namespaceSelector**
  object

  Specifies the Namespace selector to filter objects with.

  - **spec.match.namespaceSelector.excludeNames**
    array of strings

    Include all namespaces except a particular set. Support glob pattern.

  - **spec.match.namespaceSelector.labelSelector**
    object

    Specifies the label selector to filter namespaces.

    You can get more info in the documentation.

    - **spec.match.namespaceSelector.labelSelector.matchExpressions**
      array of objects

      List of label expressions for namespaces.

      Example:

      ```
      matchExpressions:
      - key: tier
        operator: In
        values:
        - production
        - staging
      ```

      - **spec.match.namespaceSelector.labelSelector.matchExpressions.key**
        string

        Required value

      - **spec.match.namespaceSelector.labelSelector.matchExpressions.operator**
        string

        Required value

        Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

      - **spec.match.namespaceSelector.labelSelector.matchExpressions.values**
        array of strings
    - **spec.match.namespaceSelector.labelSelector.matchLabels**
      object

      List of labels which a namespace should have.

      Example:

      ```
      matchLabels:
        foo: bar
        baz: who
      ```

  - **spec.match.namespaceSelector.matchNames**
    array of strings

    Include only a particular set of namespaces. Supports glob pattern.

- **spec.policies**
  object

  Required value

- **spec.policies.allowedRepos**
  array of strings

  The list of prefixes a container image is allowed to have.

  - **Element of the array**
    string

    Example:

    ```
    registry.deckhouse.io
    ```

- **spec.policies.checkContainerDuplicates**
  boolean

  Check container names and env variables for duplicates.

- **spec.policies.checkHostNetworkDNSPolicy**
  boolean

  Check `ClusterFirstWithHostNet` dnsPolicy is set for Pods with `hostNetwork: true`.

- **spec.policies.disallowedImageTags**
  array of strings

  Requires container images to have an image tag different from the ones in the specified list.

  Example:

  ```
  disallowedImageTags: latest
  ```

- **spec.policies.disallowedTolerations**
  array of objects

  Deny Pods that have tolerations from the ones in the specified list.

  See more in the Kubernetes documentation.

  - **spec.policies.disallowedTolerations.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **spec.policies.disallowedTolerations.key**
    string
  - **spec.policies.disallowedTolerations.operator**
    string

    Allowed values: `Exists`, `Equal`

  - **spec.policies.disallowedTolerations.value**
    string
- **spec.policies.imagePullPolicy**
  string

  Required image pull policy for containers.

  Allowed values: `Always`, `IfNotPresent`

- **spec.policies.ingressClassNames**
  array of strings

  List of allowed ingress class names.

- **spec.policies.maxRevisionHistoryLimit**
  integer

  A maximum value for a revision history.

- **spec.policies.priorityClassNames**
  array of strings

  List of allowed priority class names.

- **spec.policies.replicaLimits**
  object

  A range of allowed replicas. Values are inclusive.

  - **spec.policies.replicaLimits.maxReplicas**
    integer

    The maximum number of replicas allowed, inclusive.

- **spec.policies.replicaLimits.minReplicas**
  integer

  The minimum number of replicas allowed, inclusive.

- **spec.policies.requiredAnnotations**
  object

  A list of annotations and values the object must specify.

  - **spec.policies.requiredAnnotations.annotations**
    array of objects
    - **spec.policies.requiredAnnotations.annotations.allowedRegex**
      string

      If specified, a regular expression, the annotation’s value must match. The value must contain at least one match for the regular expression.

    - **spec.policies.requiredAnnotations.annotations.key**
      string

      The required annotation.

  - **spec.policies.requiredAnnotations.watchKinds**
    array of strings

    The list of kubernetes objects in the format `$apiGroup/$kind` to watch the annotations on.

    - **Element of the array**
      string

      Pattern: `^[a-z]*/[a-zA-Z]+$`

      Examples:

      ```
      apps/Deployment
      ```

      ```
      "/Pod"
      ```

      ```
      networking.k8s.io/Ingress
      ```

- **spec.policies.requiredLabels**
  object

  A list of labels and values the object must specify.

  - **spec.policies.requiredLabels.labels**
    array of objects
    - **spec.policies.requiredLabels.labels.allowedRegex**
      string

      If specified, a regular expression, the label’s value must match. The value must contain at least one match for the regular expression.

    - **spec.policies.requiredLabels.labels.key**
      string

      The required label.

  - **spec.policies.requiredLabels.watchKinds**
    array of strings

    The list of kubernetes objects in the format `$apiGroup/$kind` to watch the labels on.

    - **Element of the array**
      string

      Pattern: `^[a-z]*/[a-zA-Z]+$`

      Examples:

      ```
      apps/Deployment
      ```

      ```
      "/Pod"
      ```

      ```
      networking.k8s.io/Ingress
      ```

- **spec.policies.requiredProbes**
  array of strings

The list of probes that are required (e.g. `readinessProbe` and `livenessProbe`)

Example:

```
requiredProbes:
- readinessProbe
- livenessProbe
```

- **Element of the array**
  string

  Allowed values: `livenessProbe`, `readinessProbe`, `startupProbe`

- **spec.policies.requiredResources**
  object

  Requires containers to have defined resources set.

  - **spec.policies.requiredResources.limits**
    array of strings

    A list of limits that should be enforced (CPU, memory, or both).

    - **Element of the array**
      string

      Allowed values: `cpu`, `memory`

  - **spec.policies.requiredResources.requests**
    array of strings

    A list of requests that should be enforced (CPU, memory, or both).

    - **Element of the array**
      string

      Allowed values: `cpu`, `memory`

- **spec.policies.storageClassNames**
  array of strings

  List of allowed storage class names.

SecurityPolicyException

Scope: Namespaced
Version: v1alpha1

Describes exceptions to the cluster's security policies. Each SecurityPolicyException resource defines a set of specific allowances.

- **status**
  object
  - **status.deckhouse**
    object
    - **status.deckhouse.observed**
      object

      Contains last timestamp when the resource change was noted by the operator and its checksum.

      - **status.deckhouse.observed.checkSum**
        string

        The checksum of last observed resource.

      - **status.deckhouse.observed.lastTimestamp**
        string

        A timestamp of when the operator last noted a change to the resource.

    - **status.deckhouse.processed**
      object

      Contains last timestamp when the resource was applied in the cluster by the operator and its checksum.

      - **status.deckhouse.processed.checkSum**
        string

        The checksum of last applied resource.

      - **status.deckhouse.processed.lastTimestamp**
        string

A timestamp of when the resource was last applied in the cluster.

- **status.deckhouse.synced**
  string

  True if last observed version of the resource was successfully applied in the cluster.

- **spec**
  object

  Required value

  - **spec.network**
    object

    Exceptions for network-related parameters.

    - **spec.network.hostIPC**
      object

      Exceptions for the `hostIPC` parameter.

      - **spec.network.hostIPC.allowedValue**
        boolean

        Required value

        Allowed value for `hostIPC`.

        Default: `false`

      - **spec.network.hostIPC.metadata**
        object

        Metadata related to the exception.

    - **spec.network.hostNetwork**
      object

      Exceptions for the `hostNetwork` parameter. For pods with `hostNetwork: true`, an additional exception is performed on the ports specified in `containerPort`.

      - **spec.network.hostNetwork.allowedValue**
        boolean

        Required value

        Allowed value for `hostNetwork`.

        Default: `false`

      - **spec.network.hostNetwork.metadata**
        object

        Metadata related to the exception.

    - **spec.network.hostPID**
      object

      Exceptions for the `hostPID` parameter.

      - **spec.network.hostPID.allowedValue**
        boolean

        Required value

        Allowed value for `hostPID`.

        Default: `false`

      - **spec.network.hostPID.metadata**
        object

        Metadata related to the exception.

    - **spec.network.hostPorts**
      array of objects

      Exceptions for host ports.

      - **spec.network.hostPorts.metadata**
        object

        Metadata related to the exception.

      - **spec.network.hostPorts.port**
        integer

Allowed host port.

- **spec.network.hostPorts.protocol**
  string

  Protocol for the host port.

  Allowed values: `TCP`, `UDP`

- **spec.securityContext**
  object

  Exceptions for parameters in the `securityContext` section.

  - **spec.securityContext.allowPrivilegeEscalation**
    object

    Exceptions for the `allowPrivilegeEscalation` parameter.

    - **spec.securityContext.allowPrivilegeEscalation.allowedValue**
      boolean

      Required value

      Allowed value for `allowPrivilegeEscalation`.

      Default: `false`

    - **spec.securityContext.allowPrivilegeEscalation.metadata**
      object

      Metadata related to the exception.

  - **spec.securityContext.appArmorProfile**
    object

    List of allowed AppArmor profile values for pods and containers.

    Supported formats:

    - Via annotations: `runtime/default`, `docker/default`, `unconfined`, `localhost/some-profile.json`. `localhost/*` allows any local profile.
    - Via `securityContext`: `RuntimeDefault`, `Unconfined`, `Localhost`. For `Localhost`, specify allowed profiles using the `allowedLocalhostFiles` parameter.

    Profile types:

    - `Unconfined`: No restrictions (not recommended for security reasons).
    - `RuntimeDefault`: The default profile provided by the container runtime (such as Docker or CRI-O).
    - `Localhost`: A custom profile defined on the host and tailored for the application.

    Using `*` allows all profiles. It's not necessary to specify both formats as they are automatically mapped to each other.

    - **spec.securityContext.appArmorProfile.allowedValues**
      array of strings

      Required value

      Allowed values:

      - `runtime/default`
      - `unconfined`
      - `localhost/*` (where `*` may be a profile name or a wildcard)
      - **Element of the array**
        string

        Pattern: `^(runtime\/default|unconfined|localhost\/.+)$`

    - **spec.securityContext.appArmorProfile.metadata**
      object

      Metadata related to the exception.

  - **spec.securityContext.capabilities**
    object

    Exceptions for the `capabilities` parameter.

    - **spec.securityContext.capabilities.allowedValues**
      object

      Required value

      Allowed values for adding or dropping Linux capabilities.

        - **spec.securityContext.capabilities.allowedValues.add**
          array of strings

          List of capabilities that are allowed to be added to the container.

          Default: `[]`

        - **spec.securityContext.capabilities.allowedValues.drop**
          array of strings

          List of capabilities that are allowed to be dropped from the container.

          Default: `["all"]`

      - **spec.securityContext.capabilities.metadata**
        object

        Metadata related to the exception.

- **spec.securityContext.privileged**
  object

  Exceptions for the `privileged` parameter.

    - **spec.securityContext.privileged.allowedValue**
      boolean

      Required value

      Allowed value for `privileged`.

      Default: `false`

    - **spec.securityContext.privileged.metadata**
      object

      Metadata related to the exception.

- **spec.securityContext.procMount**
  object

  Exceptions for the `procMount` parameter.

    - **spec.securityContext.procMount.allowedValues**
      array of strings

      Required value

      List of allowed `procMount` values.

        - **Element of the array**
          string

          Allowed values: `Default`, `Unmasked`

    - **spec.securityContext.procMount.metadata**
      object

      Metadata related to the exception.

- **spec.securityContext.readOnlyRootFilesystem**
  object

  Exceptions for the `readOnlyRootFilesystem` parameter.

    - **spec.securityContext.readOnlyRootFilesystem.allowedValue**
      boolean

      Required value

      Allowed value for `readOnlyRootFilesystem`.

      Default: `false`

    - **spec.securityContext.readOnlyRootFilesystem.metadata**
      object

      Metadata related to the exception.

- **spec.securityContext.runAsNonRoot**
  object

  Exceptions for the `runAsNonRoot` parameter.

    - **spec.securityContext.runAsNonRoot.allowedValue**
      boolean

Required value

Allowed value for `runAsNonRoot`.

Default: `true`

- **spec.securityContext.runAsNonRoot.metadata**
  object

  Metadata related to the exception.

- **spec.securityContext.runAsUser**
  object

  Exceptions for the `runAsUser` parameter.

  - **spec.securityContext.runAsUser.allowedValues**
    array of integers

    Required value

    List of allowed user IDs (UIDs).

  - **spec.securityContext.runAsUser.metadata**
    object

    Metadata related to the exception.

- **spec.securityContext.seLinuxOptions**
  object

  Exceptions for the `seLinuxOptions` parameter.

  - **spec.securityContext.seLinuxOptions.allowedValues**
    array of objects

    Required value

    List of allowed SELinux configurations.

    - **spec.securityContext.seLinuxOptions.allowedValues.level**
      string

      SELinux level label.

    - **spec.securityContext.seLinuxOptions.allowedValues.role**
      string

      SELinux role label.

    - **spec.securityContext.seLinuxOptions.allowedValues.type**
      string

      SELinux type label.

    - **spec.securityContext.seLinuxOptions.allowedValues.user**
      string

      SELinux user label.

  - **spec.securityContext.seLinuxOptions.metadata**
    object

    Metadata related to the exception.

- **spec.securityContext.seccompProfile**
  object

  Exceptions for the `seccompProfile` parameter.

  - **spec.securityContext.seccompProfile.allowedValues**
    array of strings

    Required value

    List of allowed seccomp profiles.

  - **spec.securityContext.seccompProfile.metadata**
    object

    Metadata related to the exception.

- **spec.securityContext.sysctls**
  object

  Exceptions for the `sysctls` parameter.

- **spec.securityContext.sysctls.allowedValues**
  array of objects

  Required value

  List of allowed `sysctl` settings.

  - **spec.securityContext.sysctls.allowedValues.name**
    string

    Name of the `sysctl` parameter.

  - **spec.securityContext.sysctls.allowedValues.value**
    string

    Allowed value for the specified `sysctl` parameter.

- **spec.securityContext.sysctls.metadata**
  object

  Metadata related to the exception.

- **spec.volumes**
  object

  Exceptions for volume-related parameters.

  - **spec.volumes.hostPath**
    object

    Exceptions for `hostPath` volumes.

    - **spec.volumes.hostPath.allowedValues**
      array of objects

      Required value

      List of allowed `hostPath` configurations.

      - **spec.volumes.hostPath.allowedValues.metadata**
        object

        Metadata related to the exception.

        - **spec.volumes.hostPath.allowedValues.metadata.description**
          string

          Description of the exception.

      - **spec.volumes.hostPath.allowedValues.path**
        string

        Allowed `hostPath`.

      - **spec.volumes.hostPath.allowedValues.readOnly**
        boolean

        A `readOnly` value for the `hostPath`.

        Default: `false`

  - **spec.volumes.types**
    object

    Exceptions for volume types.

    - **spec.volumes.types.allowedValues**
      array of strings

      Required value

      List of allowed volume types.

    - **spec.volumes.types.metadata**
      object

      Metadata related to the exception.

SecurityPolicy

Scope: Cluster
Version: v1alpha1

Describes a security policy for a cluster.

Each `SecurityPolicy` custom resource describes rules for the objects in the cluster.

- **status**
  object
  - **status.deckhouse**
    object
    - **status.deckhouse.observed**
      object

      Contains last timestamp when the resource change was noted by the operator and its checksum.

      - **status.deckhouse.observed.checkSum**
        string

        The checksum of last observed resource.

      - **status.deckhouse.observed.lastTimestamp**
        string

        A timestamp of when the operator last noted a change to the resource.

    - **status.deckhouse.processed**
      object

      Contains last timestamp when the resource was applied in the cluster by the operator and its checksum.

      - **status.deckhouse.processed.checkSum**
        string

        The checksum of last applied resource.

      - **status.deckhouse.processed.lastTimestamp**
        string

        A timestamp of when the resource was last applied in the cluster.

    - **status.deckhouse.synced**
      string

      True if last observed version of the resource was successfully applied in the cluster.

- **spec**
  object

  Required value

  - **spec.enforcementAction**
    string

    An enforcement action as a result of the constraint:

    - `Deny` — Deny action.
    - `Dryrun` — No action. Used for debugging. Information about the event can be viewed in Grafana in dashboard `Security/Admission policy engine`.
    - `Warn` — No action; similar to `Dryrun`. Provides information about the constraint that would result in a denial if the `Deny` action is used.

    Default: `Deny`

    Allowed values: `Warn`, `Deny`, `Dryrun`

  - **spec.match**
    object

    Required value

    Container filtering rules. Use selectors to specify the pods and containers to which you want to apply the policy.

    - **spec.match.labelSelector**
      object

      Specifies the label selector to filter Pods with.

      You can get more into here.

      - **spec.match.labelSelector.matchExpressions**
        array of objects

        The list of label expressions for Pods.

        Example:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
```

- **spec.match.labelSelector.matchExpressions.key**
  string

  Required value

- **spec.match.labelSelector.matchExpressions.operator**
  string

  Required value

  Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.match.labelSelector.matchExpressions.values**
  array of strings

- **spec.match.labelSelector.matchLabels**
  object

  The list of the labels that the Pod should have.

  Example:

```
matchLabels:
  foo: bar
  baz: who
```

- **spec.match.namespaceSelector**
  object

  Specifies the Namespace selector to filter objects with.

  - **spec.match.namespaceSelector.excludeNames**
    array of strings

    Includes all namespaces except a particular set. Support glob pattern.

  - **spec.match.namespaceSelector.labelSelector**
    object

    Specifies the label selector to filter namespaces.

    You can get more info in the documentation.

    - **spec.match.namespaceSelector.labelSelector.matchExpressions**
      array of objects

      The list of label expressions for namespaces.

      Example:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
```

      - **spec.match.namespaceSelector.labelSelector.matchExpressions.key**
        string

        Required value

      - **spec.match.namespaceSelector.labelSelector.matchExpressions.operator**
        string

        Required value

        Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

      - **spec.match.namespaceSelector.labelSelector.matchExpressions.values**
        array of strings

    - **spec.match.namespaceSelector.labelSelector.matchLabels**
      object

The list of the labels that the namespace should have.

Example:

```
matchLabels:
  foo: bar
  baz: who
```

- **spec.match.namespaceSelector.matchNames**
  array of strings

  Includes only a particular set of namespaces. Supports glob pattern.

- **spec.policies**
  object

  Required value

  Policies that pods and containers must comply with.

  - **spec.policies.allowHostIPC**
    boolean

    Allows sharing the host's IPC namespace with containers.

  - **spec.policies.allowHostNetwork**
    boolean

    Allows containers to use the host's network.

  - **spec.policies.allowHostPID**
    boolean

    Allows sharing the host's PID namespace with containers.

  - **spec.policies.allowPrivilegeEscalation**
    boolean

    Allows container processes to gain more privileges than its parent process.

    By default (if not specified) — `false`. This means that if the field is omitted or explicitly set to `false`, the container is denied any privilege escalation. If you want to allow elevated privileges, set this parameter to `true`.

  - **spec.policies.allowPrivileged**
    boolean

    Allows running containers in a privileged mode.

    By default (if not specified) — `false`. This means that if the field is omitted or explicitly set to `false`, the container is denied any privilege escalation. If you want to allow elevated privileges, set this parameter to `true`.

  - **spec.policies.allowRbacWildcards**
    boolean

    Allows using wildcards (`*`) in namespaced Role and RoleBinding resources.

    ClusterRole and ClusterRoleBinding are not validated by SecurityPolicy.

    By default (if not specified) — `true`. This means that if the field is omitted or explicitly set to `true`, using wildcards is allowed. If you want to deny wildcards, set this parameter to `false`.

    Default: `true`

  - **spec.policies.allowedAppArmor**
    array of strings

    The list of AppArmor profiles the containers are permitted to use.

    Example:

    ```
    allowedAppArmor:
    - runtime/default
    - unconfined
    ```

    - **Element of the array**
      string

      An AppArmor profile.

  - **spec.policies.allowedCapabilities**
    array of strings

The list of capabilities that the containers are permitted to use.

To allow all capabilities, use `ALL`.

Example:

```
allowedCapabilities:
- SETGID
- SETUID
- NET_BIND_SERVICE
```

- **Element of the array**
  string

  A linux capability.

  Allowed values: `ALL`, `SETPCAP`, `SYS MODULE`, `SYS RAWIO`, `SYS PACCT`, `SYS ADMIN`, `SYS NICE`, `SYS RESOURCE`, `SYS TIME`, `SYS TTY CONFIG`, `MKNOD`, `AUDIT WRITE`, `AUDIT CONTROL`, `MAC OVERRIDE`, `MAC ADMIN`, `NET ADMIN`, `SYSLOG`, `CHOWN`, `NET_RAW`, `DAC OVERRIDE`, `FOWNER`, `DAC READ SEARCH`, `FSETID`, `KILL`, `SETGID`, `SETUID`, `LINUX IMMUTABLE`, `NET BIND SERVICE`, `NET BROADCAST`, `IPC LOCK`, `IPC OWNER`, `SYS_CHROOT`, `SYS_PTRACE`, `SYS_BOOT`, `LEASE`, `SETFCAP`, `WAKE_ALARM`, `BLOCK_SUSPEND`

- **spec.policies.allowedClusterRoles**
  array of strings

  A list of allowed cluster roles to bind to users.

- **spec.policies.allowedFlexVolumes**
  array of objects

  The list of Flex Volume drivers the containers are permitted to use.

  - **spec.policies.allowedFlexVolumes.driver**
    string

    A driver name.

- **spec.policies.allowedHostPaths**
  array of objects

  The list of allowed path prefixes for `hostPath` mounts. An empty list (`[]`) forbids `hostPath` for all paths. If the field is omitted, no restrictions apply.

  Example:

  ```
  allowedHostPaths:
  - pathPrefix: "/dev"
    readOnly: true
  ```

  - **spec.policies.allowedHostPaths.pathPrefix**
    string

    Required value

    The path prefix to match against the host volume.

    It does not support the `*` mask. Trailing slashes are trimmed when validating the path prefix with a host path.

    For example, the `/foo` prefix allows `/foo`, `/foo/` and `/foo/bar` path, but doesn’t allow `/food` or `/etc/foo` path.

  - **spec.policies.allowedHostPaths.readOnly**
    boolean

    When set to true, allows host volumes to be matched against the pathPrefix only if all the volume mounts are read-only.

    Default: `false`

- **spec.policies.allowedHostPorts**
  array of objects

  The list of `hostPort` ranges allowed by the rule. For pods with `hostNetwork: true`, an additional check is performed on the ports specified in `containerPort`.

  - **spec.policies.allowedHostPorts.max**
    integer

    Max value for the `hostPort`.

    - **spec.policies.allowedHostPorts.min**
      integer

      Min value for the `hostPort`.

- **spec.policies.allowedProcMount**
  string

  Allows `/proc` mount type for containers.

  Allowed values: `Default`, `Unmasked`

  Example:

  ```
  allowedProcMount: Unmasked.
  ```

- **spec.policies.allowedServiceTypes**
  array of strings

  The list of allowed service types.

  Example:

  ```
  allowedServiceTypes:
  - LoadBalancer
  - ClusterIP
  ```

    - **Element of the array**
      string

      Allowed values: `ClusterIP`, `NodePort`, `LoadBalancer`, `ExternalName`

- **spec.policies.allowedUnsafeSysctls**
  array of strings

  The list of explicitly allowed unsafe sysctls.

  To allow all unsafe sysctls, use `*`.

  Example:

  ```
  allowedUnsafeSysctls:
  - kernel.msg*
  - net.core.somaxconn
  ```

- **spec.policies.allowedVolumes**
  array of strings

  The set of the permitted volume plugins.

  Example:

  ```
  allowedVolumes:
  - hostPath
  - persistentVolumeClaim
  ```

    - **Element of the array**
      string

      Allowed values: `*`, `none`, `awsElasticBlockStore`, `azureDisk`, `azureFile`, `cephFS`, `cinder`, `configMap`, `csi`, `downwardAPI`, `emptyDir`, `ephemeral`, `fc`, `flexVolume`, `flocker`, `gcePersistentDisk`, `gitRepo`, `glusterfs`, `hostPath`, `iscsi`, `nfs`, `persistentVolumeClaim`, `photonPersistentDisk`, `portworxVolume`, `projected`, `quobyte`, `rbd`, `scaleIO`, `secret`, `storageos`, `vsphereVolume`

- **spec.policies.automountServiceAccountToken**
  boolean

  Allows pods to run with `automountServiceAccountToken` enabled.

- **spec.policies.blockWildcardDomains**
  boolean

  Block creation of Ingress objects with `*` in DNS domains.

- **spec.policies.forbiddenSysctls**
  array of strings

  The list of forbidden sysctls.

Takes precedence over allowed unsafe sysctls (allowedUnsafeSysctls).

Example:

```
forbiddenSysctls:
- kernel.msg*
- net.core.somaxconn
```

- **spec.policies.fsGroup**
  object

  Specifies which `fsGroup` values the security context is permitted to use.

  - **spec.policies.fsGroup.ranges**
    array of objects

    The list of `fsGroup` ID ranges that are allowed in `MustRunAs' mode.

    - **spec.policies.fsGroup.ranges.max**
      integer

      Max ID value.

    - **spec.policies.fsGroup.ranges.min**
      integer

      Min ID value.

  - **spec.policies.fsGroup.rule**
    string

    Required value

    Specifies the strategy of the `fsGroup` selection.

    Allowed values: `MustRunAs`, `MayRunAs`, `RunAsAny`

- **spec.policies.readOnlyRootFilesystem**
  boolean

  If set to true, only the pods with the read-only root filesystem across all containers will be permitted to run. See the Kubernetes documentation for more details.

- **spec.policies.requiredDropCapabilities**
  array of strings

  The list of capabilities that have to be dropped from the containers.

  To exclude all capabilities, use `ALL`'.

  Example:

  ```
  requiredDropCapabilities:
  - SETGID
  - SETUID
  - NET_BIND_SERVICE
  ```

  - **Element of the array**
    string

    A linux capability to drop from the containers' specs.

    Allowed values: `ALL`, `SETPCAP`, `SYS MODULE`, `SYS RAWIO`, `SYS PACCT`, `SYS ADMIN`, `SYS NICE`, `SYS RESOURCE`, `SYS TIME`, `SYS TTY CONFIG`, `MKNOD`, `AUDIT WRITE`, `AUDIT CONTROL`, `MAC OVERRIDE`, `MAC ADMIN`, `NET ADMIN`, `SYSLOG`, `CHOWN`, `NET_RAW`, `DAC OVERRIDE`, `FOWNER`, `DAC READ SEARCH`, `FSETID`, `KILL`, `SETGID`, `SETUID`, `LINUX IMMUTABLE`, `NET BIND SERVICE`, `NET BROADCAST`, `IPC LOCK`, `IPC OWNER`, `SYS_CHROOT`, `SYS_PTRACE`, `SYS_BOOT`, `LEASE`, `SETFCAP`, `WAKE_ALARM`, `BLOCK_SUSPEND`

- **spec.policies.runAsGroup**
  object

  Specifies which `runAsGroup` values the security context is permitted to use.

  - **spec.policies.runAsGroup.ranges**
    array of objects

    The list of group ID ranges that are allowed in `MustRunAs' mode.

    - **spec.policies.runAsGroup.ranges.max**
      integer

      Max ID value.

        - **spec.policies.runAsGroup.ranges.min**
          integer

          Min ID value.

      - **spec.policies.runAsGroup.rule**
        string

        Required value

        Specifies the strategy of the group ID selection.

        Allowed values: `MustRunAs`, `MayRunAs`, `RunAsAny`

- **spec.policies.runAsUser**
  object

  Specifies which `runAsUser` values the security context is permitted to use.

  - **spec.policies.runAsUser.ranges**
    array of objects

    The list of user ID ranges that are allowed in `MustRunAs' mode.

    - **spec.policies.runAsUser.ranges.max**
      integer

      Max ID value.

    - **spec.policies.runAsUser.ranges.min**
      integer

      Min ID value.

  - **spec.policies.runAsUser.rule**
    string

    Required value

    Specifies the strategy of the user ID selection.

    Allowed values: `MustRunAs`, `MustRunAsNonRoot`, `RunAsAny`

- **spec.policies.seLinux**
  array of objects

  Specifies which SElinux labels the security context is permitted to use.

  - **spec.policies.seLinux.level**
    string

    A SELinux level label that applies to the container.

  - **spec.policies.seLinux.role**
    string

    A SELinux role label that applies to the container.

  - **spec.policies.seLinux.type**
    string

    A SELinux type label that applies to the container.

  - **spec.policies.seLinux.user**
    string

    A SELinux user label that applies to the container.

- **spec.policies.seccompProfiles**
  object

  Specifies the list of allowed profiles that can be set for the Pod or container's seccomp annotations.

  - **spec.policies.seccompProfiles.allowedLocalhostFiles**
    array of strings

    Defines the local seccomp profiles (in JSON format) that can be used if `Localhost` is set in the `allowedProfiles` parameter.

    An empty list prohibits the use of any local profiles.

  - **spec.policies.seccompProfiles.allowedProfiles**
    array of strings

    The list of allowed profile values for seccomp on Pods/containers.

Both formats are supported:

- Via annotations: `runtime/default`, `docker/default`, `unconfined`, `localhost/some-profile.json`. `localhost/*` allows any local profile.
- Via securityContext: `RuntimeDefault`, `Unconfined`, `Localhost`. For `Localhost`, specify the allowed profiles using the `allowedLocalhostFiles` parameter.

Profile types:

- `Unconfined` — no restrictions (not recommended for security reasons).
- `RuntimeDefault` — the default profile provided by the container runtime (e.g., Docker, CRI-O).
- `Localhost` — a custom profile defined on the host (flexible and tailored to the application).

Using `*` allows all profiles. It's not necessary to specify both formats — they are automatically mapped to each other.

- **spec.policies.supplementalGroups**
  object

  Specifies what supplemental groups are allowed to be used by the security context.

  - **spec.policies.supplementalGroups.ranges**
    array of objects

    The list of supplemental group ID ranges that are allowed in `MustRunAs' mode.

    - **spec.policies.supplementalGroups.ranges.max**
      integer

      Max ID value.

    - **spec.policies.supplementalGroups.ranges.min**
      integer

      Min ID value.

  - **spec.policies.supplementalGroups.rule**
    string

    Required value

    Specifies the strategy of the supplemental group ID selection.

    Allowed values: `MustRunAs`, `MayRunAs`, `RunAsAny`

- **spec.policies.verifyImageSignatures**
  array of objects

  Available in editions: SE+, EE

  List of policies to verify container images signatures.

  Container images must be signed using Cosign.

  Example:

  ```
  verifyImageSignatures:
  - reference: docker.io/myrepo/*
    publicKeys:
    - |-
      -----BEGIN PUBLIC KEY-----
      .....
      -----END PUBLIC KEY-----
  - reference: company.registry.com/*
    dockerCfg: "<Base64_dockerCfg>"
    publicKeys:
    - |-
      -----BEGIN PUBLIC KEY-----
      .....
      -----END PUBLIC KEY-----
  ```

  - **spec.policies.verifyImageSignatures.ca**
    string

    A custom certificate authority to use when connecting to the container image repository.

  - **spec.policies.verifyImageSignatures.dockerCfg**
    string

A string in Base64 with authentication data for the container image repository.

If the container images are available anonymously, it is not specified.

- **spec.policies.verifyImageSignatures.publicKeys**
  array of strings

  Required value

  The list of Cosign compliant public keys.

- **spec.policies.verifyImageSignatures.reference**
  string

  Required value

  Absolute address or template for container images.

  If it contains the `*` symbol, it is considered a template for container image addresses. The `*` symbol can only appear once and only at the end. For example, for the value `company.registry.com/*`, any container images from the repository `company.registry.com` will be checked with the specified keys and access parameters.

  If it does not contain the `*` symbol, it is considered an absolute address for a container image. For example, for the value `company.registry.com/nginx`, only the nginx image from the repository `company.registry.com` will be checked with the specified keys and access parameters.

  Pattern: `^[a-z0-9\.\-:@\/]*\*?$`

  Examples:

```
reference: docker.io/myuser/*
```

```
reference: "*"
```

Custom Resources (by Gatekeeper)

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

Mutation Custom Resources

> **i** The `reinvocationPolicy: IfNeeded` is used in MutatingWebhookConfiguration. More details in the Kubernetes documentation.

Reference

Provide a configurable set of policies for modifying Kubernetes resources at the time they are deployed.

AssignMetadata

Reference

Allows you to modify the `Metadata` section of a resource. At the moment, Gatekeeper only allows **adding** `labels` and `annotations` objects. Modification of existing objects is not provided.

> **i** Using `*` in `spec.match.kinds` is not allowed. If `*` is specified, the mutation will not be applied. Instead, you must explicitly list the target resources (`kinds`) along with their corresponding `apiGroups`.

Example 1. Adding the `owner` label with the value `admin` to all namespaces:

```
apiVersion: mutations.gatekeeper.sh/v1
kind: AssignMetadata
metadata:
  name: demo-annotation-owner
spec:
  match:
    scope: Namespaced
  location: "metadata.labels.owner"
  parameters:
    assign:
      value: "admin"
```

Example 2. Adding a label in a specific namespace and only to selected resources:

```
apiVersion: mutations.gatekeeper.sh/v1
kind: AssignMetadata
metadata:
  name: set-labels-<your_namespace>
spec:
  match:
    scope: Namespaced
    namespaces: ["<your_namespace>"]
    kinds:
    - apiGroups: [""]
      kinds: ["Pod"] # The use of "*" is not allowed.
  location: "metadata.labels.<your_label_name>"
  parameters:
    assign:
      value: <your_label_value>
```

Assign

Allows you to modify fields outside the `Metadata` section.

An example of setting `imagePullPolicy` for all containers to `Always` in all namespaces except the `system` namespace:

```
apiVersion: mutations.gatekeeper.sh/v1
kind: Assign
metadata:
  name: demo-image-pull-policy
spec:
  applyTo:
  - groups: [""]
    kinds: ["Pod"]
    versions: ["v1"]
  match:
    scope: Namespaced
    kinds:
    - apiGroups: ["*"]
      kinds: ["Pod"]
    excludedNamespaces: ["system"]
  location: "spec.containers[name:*].imagePullPolicy"
  parameters:
    assign:
      value: Always
```

ModifySet

Reference

Allows you to add and remove items from a list, such as arguments for running a container. New values are added to the end of the list.

An example of removing the `--alsologtostderr` argument from all containers in a pod:

```
apiVersion: mutations.gatekeeper.sh/v1
kind: ModifySet
metadata:
  name: remove-err-logging
spec:
  applyTo:
  - groups: [""]
    kinds: ["Pod"]
    versions: ["v1"]
  location: "spec.containers[name: *].args"
  parameters:
    operation: prune
    values:
      fromList:
        - --alsologtostderr
```

AssignImage

Reference

Allows you to make changes to the `image` parameter of a resource.

An example of changing the `image` parameter to the value `my.registry.io/repo/app@sha256:abcde67890123456789abc345678901a`:

```
apiVersion: mutations.gatekeeper.sh/v1alpha1
kind: AssignImage
metadata:
  name: assign-container-image
spec:
  applyTo:
  - groups: [ "" ]
    kinds: [ "Pod" ]
    versions: [ "v1" ]
  location: "spec.containers[name:*].image"
  parameters:
    assignDomain: "my.registry.io"
    assignPath: "repo/app"
    assignTag: "@sha256:abcde67890123456789abc345678901a"
  match:
    source: "All"
    scope: Namespaced
    kinds:
    - apiGroups: [ "*" ]
      kinds: [ "Pod" ]
```

FAQ

> **i** The module lifecycle stage: General Availability

How do I configure alternative security policy management solutions?

For DKP to work correctly, extended privileges are required to run and operate system component payloads. If you are using some alternative security policy management solution (e. g., Kyverno) instead of the admission-policy-engine module, you have to configure exceptions for the following namespaces:

- `kube-system`;
- all namespaces with the `d8-*` prefix (e.g., `d8-system`).

How do I configure policy selectors?

In `OperationPolicy` and `SecurityPolicy`, the `spec.match` field determines which specific objects (pods) in the cluster the policy will apply to. It must be present in the configuration. Filtering is performed by combining two main criteria: a pod selector (`labelSelector`) and a namespace selector (`namespaceSelector`).

If both selectors are specified, the policy is applied only to those pods that **simultaneously**:

- satisfy the pod selection conditions.

- located in namespaces that passed the filtering.

If either selector is omitted, the corresponding check is not performed (all pods or namespaces will be used).

`spec.match.labelSelector` – pod selection

The `labelSelector` is used to set criteria for selecting pods by their labels. Two mutually exclusive methods are supported:

- **`matchLabels`** – a simple check for an exact label match (key-value). The pod must have all the specified labels.
- **`matchExpressions`** – flexible expressions with operators. Each expression is defined by an object with the following fields:
  - `key` (string, required) – the label name.
  - `operator` (string, required) – one of the values: `In`, `NotIn`, `Exists`, `DoesNotExist`.
  - `values` (array of strings) – a list of values for the `In` / `NotIn` operators; not specified for `Exists` / `DoesNotExist`.

All elements in the `matchExpressions` list are combined with a logical **AND** – the pod must satisfy every expression.

Examples:

```
spec:
  match:
    labelSelector:
      matchLabels:
        app: nginx
        role: frontend
```

```
spec:
  match:
    labelSelector:
      matchExpressions:
        - key: tier
          operator: In
          values:
            - production
            - staging
        - key: monitoring
          operator: Exists
```

`spec.match.namespaceSelector` – namespace selection

Allows limiting the namespaces in which the policy is active. You can use a combination of three filters:

- **`matchNames`** – an explicit list of allowed namespaces. If specified, the policy only applies to the listed namespaces.
- **`excludeNames`** – a list of excluded namespaces. The policy will apply to all namespaces **except** those specified.
- **`labelSelector`** – a selector based on the labels of the Namespace object itself. The syntax is identical to the pod selector (`matchLabels` / `matchExpressions`). It filters the namespace metadata.

All specified conditions within `namespaceSelector` are also combined with a logical **AND**. It is recommended not to mix `matchNames` and `excludeNames` without a clear need – if both are specified, the resulting set is calculated as (`matchNames` ∩ all) \ `excludeNames` (allowed ones are taken first, then excluded ones are subtracted from them).

Examples:

```
spec:
  match:
    namespaceSelector:
      matchNames:
        - production
        - staging
```

```
spec:
  match:
    namespaceSelector:
      excludeNames:
        - kube-system
        - gatekeeper-system
```

```
spec:
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          team: backend
          environment: production
```

```
spec:
  match:
    namespaceSelector:
      labelSelector:
        matchExpressions:
          - key: compliance
            operator: In
            values:
              - pci
              - sox
```

Using selectors together

The most typical scenario is limiting both by pods and by namespaces simultaneously:

```
spec:
  match:
    labelSelector:
      matchLabels:
        app: payments
        version: v2
    namespaceSelector:
      labelSelector:
        matchLabels:
          env: prod
      excludeNames:
        - legacy-prod
```

This policy will trigger for pods with labels `app=payments` and `version=v2` that are located in namespaces having the label `env=prod`, except for the `legacy-prod` namespace.

```
spec:
  match:
    labelSelector:
      matchExpressions:
        - key: security
          operator: NotIn
          values:
            - low
    namespaceSelector:
      matchNames:
        - frontend
        - backend
```

Here, the policy covers pods where the `security` label value is not equal to `low`, in the `frontend` and `backend` namespaces.

How do I extend Pod Security Standards policies?

> **i** Pod Security Standards respond to the `security.deckhouse.io/pod-policy: restricted` or `security.deckhouse.io/pod-policy: baseline` label.

To extend the Pod Security Standards policy by adding your checks to existing checks, you need to:

- Create a constraint template for the check (`ConstraintTemplate`).
- Bind it to the `restricted` or `baseline` policy.

Example of the `ConstraintTemplate` for checking a repository URL of a container image:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sallowedrepos
spec:
  crd:
    spec:
      names:
        kind: K8sAllowedRepos
      validation:
        openAPIV3Schema:
          type: object
          properties:
            repos:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.pod_security_standards.extended

        violation[{"msg": msg}] {
          container := input.review.object.spec.containers[_]
          satisfied := [good | repo = input.parameters.repos[_] ; good =
startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>, allowed repos are
%v", [container.name, container.image, input.parameters.repos])
        }

        violation[{"msg": msg}] {
          container := input.review.object.spec.initContainers[_]
          satisfied := [good | repo = input.parameters.repos[_] ; good =
startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>, allowed repos are
%v", [container.name, container.image, input.parameters.repos])
        }
```

Example of binding a check to the `restricted` policy:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: prod-repo
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaceSelector:
      matchLabels:
        security.deckhouse.io/pod-policy: restricted
  parameters:
    repos:
      - "mycompany.registry.com"
```

The example demonstrates the configuration of checking the repository address in the `image` field for all Pods created in the namespace having the `security.deckhouse.io/pod-policy : restricted` label. A Pod will not be created if the address in the `image` field of the Pod does not start with `mycompany.registry.com`.

The Gatekeeper documentation may find more info about templates and policy language.

Find more examples of checks for policy extension in the Gatekeeper Library.

How to allow some Pod Security Standards policies without disabling whole list?

To apply only the required security policies without turning off the entire built-in set:

1. Add the `security.deckhouse.io/pod-policy: privileged` label to your namespace in order to disable built-in policies.
2. Create a SecurityPolicy that matches the baseline or restricted policy while also editing the list of `policies` elements as you see fit.
3. Add a label to your namespace that matches the `namespaceSelector` in the SecurityPolicy. In the examples below, the label is `security-policy.deckhouse.io/baseline-enabled: "true"` or `security-policy.deckhouse.io/restricted-enabled: "true"`.

> **i** The `allowedHostPaths` field defines the list of allowed path prefixes for `hostPath` mounts. An empty list (`[]`) forbids `hostPath` for all paths. If the field is omitted, no restrictions apply.

SecurityPolicy that matches baseline:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: baseline
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: false
    allowHostNetwork: false
    allowHostPID: false
    allowPrivilegeEscalation: true
    allowPrivileged: false
    allowedAppArmor:
      - runtime/default
      - localhost/*
    allowedCapabilities:
      - AUDIT_WRITE
      - CHOWN
      - DAC_OVERRIDE
      - FOWNER
      - FSETID
      - KILL
      - MKNOD
      - NET_BIND_SERVICE
      - SETFCAP
      - SETGID
      - SETPCAP
      - SETUID
      - SYS_CHROOT
    allowedHostPaths: []
    allowedHostPorts:
      - max: 0
        min: 0
    allowedProcMount: Default
    allowedUnsafeSysctls:
      - kernel.shm_rmid_forced
      - net.ipv4.ip_local_port_range
      - net.ipv4.ip_unprivileged_port_start
      - net.ipv4.tcp_syncookies
      - net.ipv4.ping_group_range
      - net.ipv4.ip_local_reserved_ports
      - net.ipv4.tcp_keepalive_time
      - net.ipv4.tcp_fin_timeout
      - net.ipv4.tcp_keepalive_intvl
      - net.ipv4.tcp_keepalive_probes
    seLinux:
      - type: ""
      - type: container_t
      - type: container_init_t
      - type: container_kvm_t
      - type: container_engine_t
    seccompProfiles:
      allowedProfiles:
        - RuntimeDefault
        - Localhost
        - undefined
        - ''
      allowedLocalhostFiles:
        - '*'
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          security-policy.deckhouse.io/baseline-enabled: "true"
```

`SecurityPolicy` that matches restricted standard:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: restricted
spec:
  enforcementAction: Deny
  policies:
    allowHostIPC: false
    allowHostNetwork: false
    allowHostPID: false
    allowPrivilegeEscalation: false
    allowPrivileged: false
    allowedAppArmor:
      - runtime/default
      - localhost/*
    allowedCapabilities:
      - NET_BIND_SERVICE
    allowedHostPaths: []
    allowedHostPorts:
      - max: 0
        min: 0
    allowedProcMount: Default
    allowedUnsafeSysctls:
      - kernel.shm_rmid_forced
      - net.ipv4.ip_local_port_range
      - net.ipv4.ip_unprivileged_port_start
      - net.ipv4.tcp_syncookies
      - net.ipv4.ping_group_range
      - net.ipv4.ip_local_reserved_ports
      - net.ipv4.tcp_keepalive_time
      - net.ipv4.tcp_fin_timeout
      - net.ipv4.tcp_keepalive_intvl
      - net.ipv4.tcp_keepalive_probes
    allowedVolumes:
      - configMap
      - csi
      - downwardAPI
      - emptyDir
      - ephemeral
      - persistentVolumeClaim
      - projected
      - secret
    requiredDropCapabilities:
      - ALL
    runAsUser:
      rule: MustRunAsNonRoot
    seLinux:
      - type: ""
      - type: container_t
      - type: container_init_t
      - type: container_kvm_t
      - type: container_engine_t
    seccompProfiles:
      allowedProfiles:
        - RuntimeDefault
        - Localhost
      allowedLocalhostFiles:
        - '*'
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          security-policy.deckhouse.io/restricted-enabled: "true"
```

What if there are multiple policies (operational or security) that are applied to the same object?

In that case the object’s specification have to fulfil all the requirements imposed by the policies.

For example, consider the following two security policies:

```
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: foo
spec:
  enforcementAction: Deny
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          name: test
  policies:
    readOnlyRootFilesystem: true
    requiredDropCapabilities:
    - MKNOD
---
apiVersion: deckhouse.io/v1alpha1
kind: SecurityPolicy
metadata:
  name: bar
spec:
  enforcementAction: Deny
  match:
    namespaceSelector:
      labelSelector:
        matchLabels:
          name: test
  policies:
    requiredDropCapabilities:
    - NET_BIND_SERVICE
```

Then, in order to fulfill the requirements of the above security policies, the following settings must be set in a container specification:

```
    securityContext:
      capabilities:
        drop:
          - MKNOD
          - NET_BIND_SERVICE
      readOnlyRootFilesystem: true
```

Verification of image signatures

The module implements a function for verifying signatures of container images signed using Cosign. For more details on signing and verifying container images, see the DKP documentation.

How to block deleting a node without a label

> **i** DELETE operations are handled by Gatekeeper by default.

You can create your own Gatekeeper policy to block Node deletion unless a special label is present. The example below uses `oldObject` to check labels on the Node being deleted:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: d8customnodedeleteguard
spec:
  crd:
    spec:
      names:
        kind: D8CustomNodeDeleteGuard
      validation:
        openAPIV3Schema:
          type: object
          properties:
            requiredLabelKey:
              type: string
            requiredLabelValue:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.custom

        is_delete { input.review.operation == "DELETE" }
        is_node { input.review.kind.kind == "Node" }

        has_required_label {
          key := input.parameters.requiredLabelKey
          val := input.parameters.requiredLabelValue
          obj := input.review.oldObject
          obj.metadata.labels[key] == val
        }

        violation[{"msg": msg}] {
          is_delete
          is_node
          not has_required_label
          msg := sprintf("Node deletion is blocked. Add label %q=%q to proceed.",
[input.parameters.requiredLabelKey, input.parameters.requiredLabelValue])
        }
---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: D8CustomNodeDeleteGuard
metadata:
  name: require-node-delete-label
spec:
  enforcementAction: warn
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Node"]
  parameters:
    requiredLabelKey: "admission.deckhouse.io/allow-delete"
    requiredLabelValue: "true"
```

How to deny kubectl exec and kubectl attach to specific Pods?

i The `admission-policy-engine` module's webhook routes `CONNECT` requests for `pods/exec` and `pods/attach` through Gatekeeper. This allows custom policies to allow or deny `kubectl exec` and `kubectl attach` operations.

Built-in policy for heritage: deckhouse Pods

To protect system components managed by Deckhouse, the `admission-policy-engine` module includes a built-in policy `D8DenyExecHeritage` that forbids running `kubectl exec` and `kubectl attach` operations to all Pods with the `heritage: deckhouse` label.

This policy doesn't apply to the following users who are allowed to run `kubectl exec` and `kubectl attach` operations to Pods labeled with `heritage: deckhouse`:

- `system:sudouser`;
- service accounts from `d8-*` namespaces (`system:serviceaccount:d8-*`);
- service accounts from `kube-*` namespaces (`system:serviceaccount:kube-*`).

Custom policy example

You can create your own Gatekeeper policy to deny `kubectl exec` and `kubectl attach` operations in specific namespaces. In the following example, `input.review.operation` and `input.review.resource.resource` are used to check for `CONNECT` operations:

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: d8customdenyexec
spec:
  crd:
    spec:
      names:
        kind: D8CustomDenyExec
      validation:
        openAPIV3Schema:
          type: object
          properties:
            forbiddenNamespaces:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package d8.custom

        is_connect {
          input.review.operation == "CONNECT"
        }

        # requestSubResource is preferred, but fall back to subResource for older APIs
        subresource_is(sub) {
          sr := object.get(input.review, "requestSubResource", input.review.subResource)
          sr == sub
        }

        is_exec_or_attach {
          input.review.resource.resource == "pods"
          subresource_is("exec")
        }

        is_exec_or_attach {
          input.review.resource.resource == "pods"
          subresource_is("attach")
        }

        is_forbidden_namespace {
          ns := input.review.namespace
          ns == input.parameters.forbiddenNamespaces[_]
        }

        violation[{"msg": msg}] {
          is_connect
          is_exec_or_attach
          is_forbidden_namespace
          msg := sprintf("Exec/attach is forbidden in namespace %q", [input.review.namespace])
        }
---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: D8CustomDenyExec
metadata:
```

```
  name: deny-exec-in-namespaces
spec:
  enforcementAction: deny
  match:
    kinds:
      - apiGroups: ["*"]
        kinds: ["*"]
    scope: Namespaced
  parameters:
    forbiddenNamespaces:
      - production
      - staging
```

Key data and checks for CONNECT validation:

- Use `input.review.operation == "CONNECT"` to check for `CONNECT` operations.
- User information is available in `input.review.userInfo.username` and `input.review.userInfo.groups`.
- The namespace is available in `input.review.namespace`.

The cert-manager module

> **i** The module lifecycle stage: General Availability

This module installs the reliable and highly available cert-manager release 1.20.0.

The installation process automatically takes into account cluster aspects:

- the component (webhook) that the `kube-apiserver` is accessing is installed on master nodes;
- if the webhook is unavailable, the `apiservice` is temporary deleted so that the unavailability of *cert-manager* does not block regular cluster operation.

The module itself is updated automatically (including the migration of cert-manager resources).

Features of the cert-manager module (with the changes made)

The module has all the features of the original cert-manager, including:

- Provisioning certificates of all the supported CA such as *Let's Encrypt*, *HashiCorp Vault*, *Venafi*;
- Issuing self-signed certificates;
- Keeping certificates up-to-date, reissuing them automatically, etc.

Changes to the original cert-manager were made so that the `cm-acme-http-solver` Pods could run on master and dedicated nodes.

Monitoring

The module can expose metrics in the Prometheus format, allowing you to monitor:

- certificate validity;
- correctness of the certificate reissue.

Module roles

The module has several well-thought-out roles for managing resources:

- `User` – has read-only access to Certificate & Issuers resources in the permitted namespaces and to the global clusterIssues;
- `Editor` – manages Certificate and Issuer resources in the permitted namespaces;
- `ClusterEditor` – manages Certificate & Issuer resources in all namespaces;
- `SuperAdmin` – manages internal service objects.

Third-party components

List of third-party software used in the `cert-manager` module:

- cert-manager 1.20.0

  License: Apache License 2.0

  A Kubernetes add-on to automate the management and issuance of TLS certificates from various issuing sources.

Configuration

> **i** The module lifecycle stage: General Availability

The module does not have any mandatory parameters.

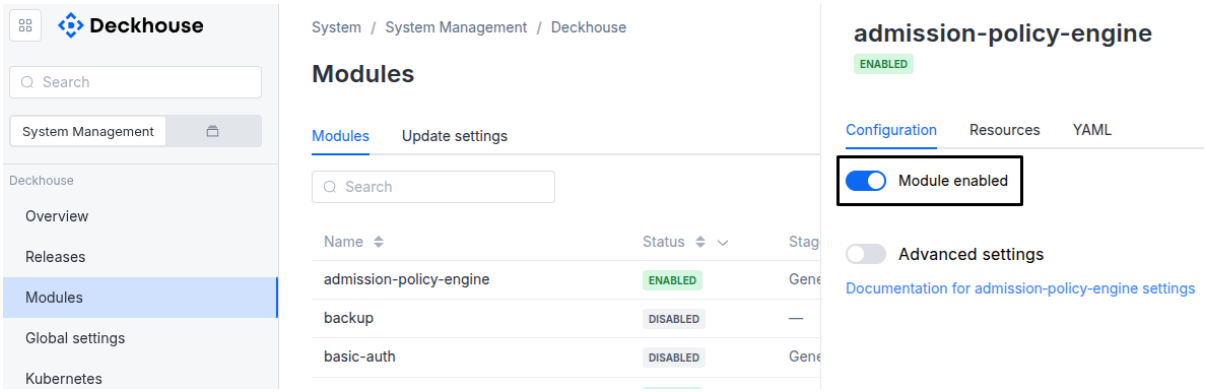
The module has 6 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cert-manager` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cert-manager
```

- **Using ModuleConfig `cert-manager`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cert-manager` (create it if necessary);

  Example of a manifest to enable module `cert-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cert-manager
spec:
  enabled: true
```

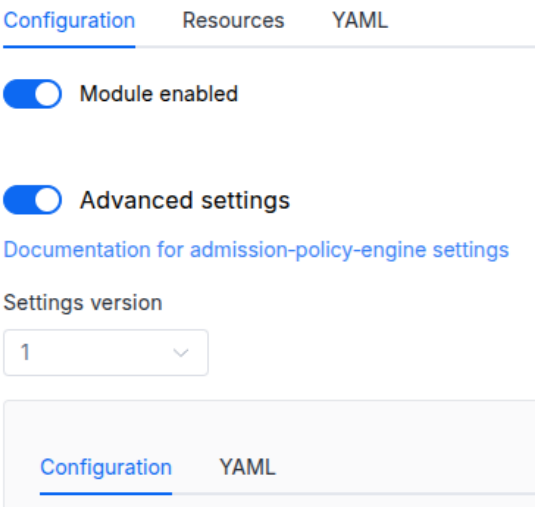
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cert-manager` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `cert-manager` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cert-manager`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cert-manager` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cert-manager
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cert-manager` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cert-manager`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cert-manager
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.cleanupOrphanSecrets**
    boolean

    Delete a secret with a certificate automatically if the corresponding Certificate resource was deleted from the cluster.

    Default: `false`

    Examples:

    ```
    cleanupOrphanSecrets: true
    ```

    ```
    cleanupOrphanSecrets: false
    ```

- **settings.cloudDNSServiceAccount**
  string

  The Service Account for Google Cloud for the same project that has the DNS Administrator role.

  Example:

  ```
  cloudDNSServiceAccount:
  eyJzYSI6ICJhYmNkZWZnaEBzZXJ2aWNlYWNjb3VudC5jb20iLCAicHJvamVjdF9pZCI6ImFhYWFhIn0=
  ```

- **settings.cloudflareAPIToken**
  string

  API Tokens allow application-scoped keys bound to specific DNS zones.

  API Tokens are recommended for higher security, since they have more restrictive permissions and are more easily revocable.

  It allows you to verify that domains specified in the Certificate resource are managed by `cert-manager` and kept by the Cloudflare DNS provider. Verification is performed by adding special TXT records for the ACME DNS01 Challenge Provider domain.

  Example:

  ```
  cloudflareAPIToken: token
  ```

- **settings.cloudflareEmail**
  string

  The email used for accessing the Cloudflare platform.

  Example:

  ```
  cloudflareEmail: example@example.com
  ```

- **settings.cloudflareGlobalAPIKey**
  string

  The Cloudflare Global API key for managing DNS records

  It allows you to verify that domains specified in the Certificate resource are managed by `cert-manager` and kept by the Cloudflare DNS provider.

  Verification is performed by adding special TXT records for the ACME DNS01 Challenge Provider domain.

  Example:

  ```
  cloudflareGlobalAPIKey: key
  ```

- **settings.digitalOceanCredentials**
  string

  The Access Token for the Digital Ocean API (you can create it in the `API` section).

  Example:

  ```
  digitalOceanCredentials: creds
  ```

- **settings.disableLetsencrypt**
  boolean

  Disable `letsencrypt` and `letsencrypt-staging` ClusterIssuer objects (if set to `true`).

  Examples:

  ```
  disableLetsencrypt: true
  ```

  ```
  disableLetsencrypt: false
  ```

- **settings.email**
  string

  The email used for sending notifications by LetsEncrypt.

  Example:

```
email: example@example.com
```

- **settings.enableCAInjector**
  boolean

  Enable CAInjector. It only needs to inject CA certs into `ValidatingWebhookConfiguration`, `MutatingWebhookConfiguration`, `CustomResourceDefinition` and `APIService`. Deckhouse does not use CAInjector, so you have to enable it only if you use custom CA injections in your services.

  Default: `false`

  Examples:

  ```
  enableCAInjector: true
  ```

  ```
  enableCAInjector: false
  ```

- **settings.enableListenerSet**
  boolean

  Enable ListenerSet support in cert-manager by turning on the `ListenerSets` feature gate.

  Additionally, when Gateway API integration is enabled, this flag also enables ListenerSet integration for Gateway API.

  Default: `false`

  Examples:

  ```
  enableListenerSet: true
  ```

  ```
  enableListenerSet: false
  ```

- **settings.ingressClassHttp01**
  string

  The name of the `ingressClass` used to confirm ownership of domain using the ACME HTTP-01 challenges method. If the parameter is omitted, the default `ingressClass` is used.

  Example:

  ```
  ingressClassHttp01: nginx
  ```

- **settings.letsencrypt**
  object

  Settings for the `letsencrypt` ClusterIssuer.

  - **settings.letsencrypt.resources**
    object

    Resource requests and limits (CPU and memory) for the workload using the `letsencrypt` ClusterIssuer.

    Example:

    ```
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 256Mi
    ```

    - **settings.letsencrypt.resources.limits**
      object
      - **settings.letsencrypt.resources.limits.cpu**
        string

        Required value

        The same as in the pods' `spec.containers.resources.limits.cpu` parameter in Kubernetes.

        Default: `50m`

        Pattern: `^[0-9]+m?$`

        - **settings.letsencrypt.resources.limits.memory**
          string or number

          Required value

          The same as in the pods' `spec.containers.resources.limits.memory` parameter in Kubernetes.

          Default: `64Mi`

          Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

      - **settings.letsencrypt.resources.requests**
        object

        Resource requests and limits (CPU and memory) for pods that create the `letsencrypt` ClusterIssuer for http01 challenges

        - **settings.letsencrypt.resources.requests.cpu**
          string

          Required value

          The same as in the pods' `spec.containers.resources.requests.cpu` parameter in Kubernetes.

          Default: `10m`

          Pattern: `^[0-9]+m?$`

        - **settings.letsencrypt.resources.requests.memory**
          string or number

          Required value

          The same as in the pods' `spec.containers.resources.requests.memory` parameter in Kubernetes.

          Default: `32Mi`

          Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.letsencryptStaging**
    object

    Settings for the `letsencrypt-staging` ClusterIssuer.

    - **settings.letsencryptStaging.resources**
      object

      Resource requests and limits (CPU and memory) for pods that create the `letsencrypt-staging` ClusterIssuer for http01 challenges.

      Example:

      ```
      resources:
        requests:
          cpu: 100m
          memory: 128Mi
        limits:
          cpu: 250m
          memory: 256Mi
      ```

      - **settings.letsencryptStaging.resources.limits**
        object
        - **settings.letsencryptStaging.resources.limits.cpu**
          string

          Required value

          The same as in the pods' `spec.containers.resources.limits.cpu` parameter in Kubernetes.

          Default: `50m`

          Pattern: `^[0-9]+m?$`

        - **settings.letsencryptStaging.resources.limits.memory**
          string or number

          Required value

          The same as in the pods' `spec.containers.resources.limits.memory` parameter in

Kubernetes.

Default: `64Mi`

Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.letsencryptStaging.resources.requests**
  object
  - **settings.letsencryptStaging.resources.requests.cpu**
    string

    Required value

    The same as in the pods' `spec.containers.resources.requests.cpu` parameter in Kubernetes.

    Default: `10m`

    Pattern: `^[0-9]+m?$`
  - **settings.letsencryptStaging.resources.requests.memory**
    string or number

    Required value

    The same as in the pods' `spec.containers.resources.requests.memory` parameter in Kubernetes.

    Default: `32Mi`

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.maxConcurrentChallenges**
  integer

  The maximum number of challenges that can be scheduled as 'processing' at once. (default 60)

  Allowed values: `0 <= X`

  Example:

  ```
  maxConcurrentChallenges: 25
  ```

- **settings.nodeSelector**
  object

  The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  nodeSelector:
    has-gpu: 'true'
  ```

- **settings.recursiveSettings**
  object

  Parameters for using recursive DNS servers, which are used to check the existence of a DNS record, before initiating the domain ownership verification process using the ACME DNS-01 method.

  Example:

  ```
  recursiveSettings:
    nameservers:
    - 8.8.8.8:53
    - https://1.1.1.1
    useOnlyRecursive: true
  ```

  - **settings.recursiveSettings.nameservers**
    array of strings

    List of recursive DNS servers (IP:port or HTTPS URLs).

    - **Element of the array**
      string

      Pattern: `^(?:[0-9]{1,3}(?:\.[0-9]{1,3}){3}:\d{1,5}|https?://[^\s]+)$`
  - **settings.recursiveSettings.useOnlyRecursive**
    boolean

When set to true, only the specified recursive DNS servers will be used to check for the existence of a record in the DNS before starting the domain ownership verification process.

Default: `false`

- **settings.route53AccessKeyID**
  string

  The Access Key ID of the user with the attached Amazon Route53 IAM Policy for managing domain records.

  Example:

  ```
  route53AccessKeyID: key_id
  ```

- **settings.route53SecretAccessKey**
  string

  The Secret Access Key of the user with privileges to manage domain records.

  Example:

  ```
  route53SecretAccessKey: secret
  ```

- **settings.tolerations**
  array of objects

  The same as in the pods' `spec.tolerations` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  tolerations:
  - key: dedicated.deckhouse.io
    operator: Equal
    value: cert-manager
  ```

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

The module uses the standard cert-manager Custom Resource called Certificate to request certificates.

Usage

> **i** The module lifecycle stage: General Availability

An example of provisioning a certificate

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com                      # the name of the certificate; you can use it to view
the cert's status
  namespace: default
spec:
  secretName: example-com-tls              # the name of the secret to store a private key and
a certificate
  issuerRef:
    kind: ClusterIssuer                  # the link to the certificate "issuer", see more
below
    name: letsencrypt
  commonName: example.com                 # the main certificate domain
  dnsNames:                              # additional domains (At least one DNS Name or IP
address is required)
  - www.example.com
  - admin.example.com
```

Here:

- a separate Ingress resource is created for the duration of the challenge (thus, authentication and whitelist of the primary Ingress will not interfere with the process),
- you can issue a single certificate for several Ingress resources (the deletion of the resource based on the `tls-acme` annotation won't affect it in any way),
- you can issue a certificate with multiple DNS names (as in the example above),
- you can validate different domains that are part of the same certificate using different Ingress controllers.

Read more in the cert-manager documentation.

Issuing a DNS wildcard certificate using Cloudflare

1. Get the `Global API Key` and `Email Address`:
   - Go to https://dash.cloudflare.com/profile.
   - You can find an active `Email Address` at the very top of the page.
   - Click the `View` button at the bottom of the page next to the `Global API Key`.

   You will see the key for interacting with the Cloudflare API (as well as the account email).

2. Edit the cert-manager module configuration and add the following parameters:

   ```
   settings:
     cloudflareGlobalAPIKey: APIkey
     cloudflareEmail: some@mail.somedomain
   ```

   or

   ```
   settings:
     cloudflareAPIToken: some-token
     cloudflareEmail: some@mail.somedomain
   ```

   After that, Deckhouse will automatically create ClusterIssuer and Secret for Cloudflare in the `d8-cert-manager` namespace.

   - Configuration with APIToken is more secure and recommended for use.

3. Create a Certificate with validation via Cloudflare. Note that you must specify `cloudflareGlobalAPIKey` and `cloudflareEmail` in Deckhouse beforehand:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: domain-wildcard
  namespace: app-namespace
spec:
  secretName: tls-wildcard
  issuerRef:
    name: cloudflare
    kind: ClusterIssuer
  commonName: "*.domain.com"
  dnsNames:
  - "*.domain.com"
```

4. Create an Ingress:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: domain-wildcard
  namespace: app-namespace
spec:
  ingressClassName: nginx
  rules:
  - host: "*.domain.com"
    http:
      paths:
      - backend:
          service:
            name: svc-web
            port:
              number: 80
        path: /
  tls:
  - hosts:
    - "*.domain.com"
    secretName: tls-wildcard
```

Issuing a DNS wildcard certificate using Route53

1. Create a user with the appropriate permissions.
   - For this, go to the policy management page and create a policy as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "route53:GetChange",
            "Resource": "arn:aws:route53:::change/*"
        },
        {
            "Effect": "Allow",
            "Action": "route53:ChangeResourceRecordSets",
            "Resource": "arn:aws:route53:::hostedzone/*"
        },
        {
            "Effect": "Allow",
            "Action": "route53:ListHostedZonesByName",
            "Resource": "*"
        }
    ]
}
```

   - Go to the user management page and create a user with the above policy.

2. Edit the cert-manager module configuration and add the following parameters:

   ```
   settings:
     route53AccessKeyID: <ACCESS_KEY_ID>
     route53SecretAccessKey: <SECRET_ACCESS_KEY>
   ```

   After that, Deckhouse will automatically create ClusterIssuer and Secret for route53 in the `d8-cert-manager` namespace.

3. Create a Certificate with validation via route53. Note that you must specify `route53AccessKeyID` and `route53SecretAccessKey` in Deckhouse beforehand:

   ```
   apiVersion: cert-manager.io/v1
   kind: Certificate
   metadata:
     name: domain-wildcard
     namespace: app-namespace
   spec:
     secretName: tls-wildcard
     issuerRef:
       name: route53
       kind: ClusterIssuer
     commonName: "*.domain.com"
     dnsNames:
     - "*.domain.com"
   ```

Issuing a DNS wildcard certificate using Google

1. Create a service account with the appropriate role:
   - Go to the policy management page.
   - Select your project.
   - Create a service account with the desired name (e.g., `dns01-solver`).
   - Switch to the service account created.
   - Add a key by clicking the “Add key” button.
   - The `.json` file with the key data will be saved to your computer.
   - Encode the resulting file using the **base64** algorithm:

     ```
     base64 project-209317-556c656b81c4.json
     ```

2. Use the resulting **base-64** string for setting the `cloudDNSServiceAccount` module parameter.

   After that, Deckhouse will automatically create ClusterIssuer and Secret for cloudDNS in the `d8-cert-manager` namespace.

3. Create a Certificate with validation via cloudDNS:

   ```
   apiVersion: cert-manager.io/v1
   kind: Certificate
   metadata:
     name: domain-wildcard
     namespace: app-namespace
   spec:
     secretName: tls-wildcard
     issuerRef:
       name: clouddns
       kind: ClusterIssuer
     dnsNames:
     - "*.domain.com"
   ```

Issuing a self-signed certificate

In this case, the entire process is even more straightforward than that of LetsEncrypt. Simply replace the issuer name (`letsencrypt`) with `selfsigned`:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com                     # Name of the certificate; you can use it to view the
cert's status.
  namespace: default
spec:
  secretName: example-com-tls             # Name of the secret to store a private key and a
certificate.
  issuerRef:
    kind: ClusterIssuer                  # Link to the ClusterIssuer.
    name: selfsigned
  commonName: example.com                # Main certificate domain.
  dnsNames:                             # Additional domain certificates. Requires at least a
duplicate commonName record.
  - www.example.com
  - admin.example.com
```

i An example of creating a self-signed certificate manually, without using the `cert-manager`, is available in FAQ.

FAQ

i The module lifecycle stage: General Availability

What types of certificates are supported?

The module installs the following ClusterIssuers:

- `letsencrypt`
- `letsencrypt-staging`
- `selfsigned`
- `selfsigned-no-trust`

If you need support for other types of certificates, you can add them yourself.

How to add an additional `ClusterIssuer`?

When is an additional `ClusterIssuer` required?

The standard delivery set includes `ClusterIssuer` that issues certificates from the trusted public certificate authority Let’s Encrypt or issues self-signed certificates.

To issue certificates for a domain name via Let’s Encrypt, the service requires that you verify domain ownership. The `cert-manager` supports several methods for verifying domain ownership when using `ACME` (Automated Certificate Management Environment):

- `HTTP-01` - when using this method `cert-manager` will create a temporary Pod in the cluster that will listen on a specific URL to verify domain ownership. For it to work, you must be able to direct external traffic to this Pod, usually via `Ingress`.
- `DNS-01` - when using this method `cert-manager` makes a TXT record in DNS to verify domain ownership. The `cert-manager` has built-in support for popular DNS providers: AWS Route53, Google Cloud DNS, Cloudflare, and others.

✖ The `HTTP-01` method does not support issuing wildcard certificates.

The `ClusterIssuers` in standard delivery set that issue certificates via Let’s Encrypt are divided into two types:

1. `ClusterIssuer` specific to the cloud provider being used.

   Added automatically when filling in the module settings associated with the cloud provider.
   These `ClusterIssuers` support the `DNS-01` method.

   - `clouddns`
   - `cloudflare`
   - `digitalocean`
   - `route53`

2. `ClusterIssuers` using the `HTTP-01` method.

   Added automatically unless their creation is disabled in the module settings.

   - `letsencrypt`
   - `letsencrypt-staging`

In this way, an additional `ClusterIssuer` may be required in the following cases:

1. Certificates need to be issued in a CA other than Let's Encrypt (including a private one). Supported CAs are available in the `cert-manager` documentation
2. Certificates need to be issued via Let's Encrypt using the `DNS-01` method via a third-party provider.

How to add an additional `ClusterIssuer` using Let's Encrypt and `DNS-01` verification method?

To verify domain ownership via Let's Encrypt using the `DNS-01` method, you need to configure the ability to create TXT records in a public DNS.

`cert-manager` has support for mechanisms for creating TXT records in some popular DNS: `AzureDNS`, `Cloudflare`, `Google Cloud DNS`, etc.
The full list is available in the `cert-manager` documentation.

The module automatically creates `ClusterIssuer` of supported cloud providers when filling in the module settings related to the cloud used.
If necessary, you can create such `ClusterIssuer` yourself.

An example of using AWS Route53 is available in the section How to protect `cert-manager` credentials.
The list of all possible `ClusterIssuer`s that can be created is available in the module templates.

Using third-party DNS providers is implemented via the `webhook` method.

When cert-manager makes an `ACME DNS-01` call, it sends a request to the webhook server, which then performs the necessary operations to update the DNS record.
When using this method, you need to place a service that will process the hook and create a TXT record in the DNS provider.

As an example, let's consider using the `Yandex Cloud DNS` service.

1. To process the webhook, first place the `Yandex Cloud DNS ACME webhook` service in the cluster according to the official documentation
2. Then, create the `ClusterIssuer` resource:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: yc-clusterissuer
  namespace: default
spec:
  acme:
    # You must replace this email address with your own.
    # Let's Encrypt will use this to contact you about expiring
    # certificates, and issues related to your account.
    email: your@email.com
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      # Secret resource that will be used to store the account's private key.
      name: secret-ref
    solvers:
      - dns01:
          webhook:
            config:
              # The ID of the folder where dns-zone located in
              folder: <your folder ID>
              # This is the secret used to access the service account
              serviceAccountSecretRef:
                name: cert-manager-secret
                key: iamkey.json
            groupName: acme.cloud.yandex.com
            solverName: yandex-cloud-dns
```

How to add an additional `Issuer` and `ClusterIssuer` using HashiCorp Vault to issue certificates?

You can use this manual for configuring certificate issuance using Vault.

After configuring PKI and enabling Kubernetes authorization, follow these steps:

- Create a `ServiceAccount`:

```
d8 k create serviceaccount issuer
```

- Create a `Secret` with a token for this service account. Starting with Kubernetes 1.24, the secret for a `ServiceAccount` is not created automatically:

```
d8 k apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: issuer
  annotations:
    kubernetes.io/service-account.name: issuer
type: kubernetes.io/service-account-token
EOF
```

- Create an Issuer:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: vault-issuer
  namespace: default
spec:
  vault:
    # HashiCorp instruction has mistype here
    server: http://vault.default.svc.cluster.local:8200
    path: pki/sign/example-dot-com # configure in pki setup step
    auth:
      kubernetes:
        mountPath: /v1/auth/kubernetes
        role: issuer
        secretRef:
          name: issuer
          key: token
EOF
```

- Create a Certificate resource, to get a TLS certificate, which is issued by Vault CA:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com
  namespace: default
spec:
  secretName: example-com-tls
  issuerRef:
    name: vault-issuer
  # domains are set on PKI setup
  commonName: www.example.com
  dnsNames:
  - www.example.com
EOF
```

How to add `ClusterIssuer` using its own or intermediate CA to to issue certificates?

Follow the steps below to use a custom or interim CA:

- Generate a certificate (if necessary):

```
openssl ecparam -name prime256v1 -genkey -out rootCAKey.pem
openssl req -x509 -sha256 -new -nodes -key rootCAKey.pem -days 3650 -out rootCACert.pem
```

- In the `d8-cert-manager` namespace, create a secret containing certificate file data.

  An example of creating a secret with d8 k:

  ```
  d8 k create secret tls internal-ca-key-pair -n d8-cert-manager --key="rootCAKey.pem" --
  cert="rootCACert.pem"
  ```

  An example of creating a secret from a YAML file (the contents of the certificate files must be Base64-encoded):

  ```
  apiVersion: v1
  data:
    tls.crt: <OUTPUT OF `cat rootCACert.pem | base64 -w0`>
    tls.key: <OUTPUT OF `cat rootCAKey.pem | base64 -w0`>
  kind: Secret
  metadata:
    name: internal-ca-key-pair
    namespace: d8-cert-manager
  type: Opaque
  ```

  You can use any name you like for the secret.

- Create a `ClusterIssuer` using the secret you created earlier:

  ```
  apiVersion: cert-manager.io/v1
  kind: ClusterIssuer
  metadata:
    name: inter-ca
  spec:
    ca:
      secretName: internal-ca-key-pair   # Name of the secret you created earlier.
  ```

  You can use any name as your `ClusterIssuer` name.

You can now use the created `ClusterIssuer` to issue certificates for all Deckhouse components or a particular component.

For example, to issue certificates for all Deckhouse components, specify the `ClusterIssuer` name in the ClusterIssuerName global parameter (`d8 k edit mc global`):

```
spec:
  settings:
    modules:
      https:
        certManager:
          clusterIssuerName: inter-ca
        mode: CertManager
      publicDomainTemplate: '%s.<public_domain_template>'
  version: 1
```

How to secure `cert-manager` credentials?

If you don’t want to store credentials in the Deckhouse configuration (security reasons, for example), feel free to create your own ClusterIssuer / Issuer. For example, you can create your own `ClusterIssuer` for a route53 service in this way:

- Create a Secret with credentials:

  ```
  d8 k apply -f - <<EOF
  apiVersion: v1
  kind: Secret
  type: Opaque
  metadata:
    name: route53
    namespace: default
  data:
    secret-access-key: MY-AWS-ACCESS-KEY-TOKEN
  EOF
  ```

- Create a simple `ClusterIssuer` with reference to that secret:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: route53
  namespace: default
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: route53-tls-key
    solvers:
    - dns01:
        route53:
          region: us-east-1
          accessKeyID: MY-AWS-ACCESS-KEY-ID
          secretAccessKeySecretRef:
            name: route53
            key: secret-access-key
EOF
```

- Order certificates as usual, using created `ClusterIssuer`:

```
d8 k apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com
  namespace: default
spec:
  secretName: example-com-tls
  issuerRef:
    name: route53
  commonName: www.example.com
  dnsNames:
  - www.example.com
EOF
```

Does the tls-acme annotation work?

Yes, it works! The dedicated component (`cert-manager-ingress-shim`) automatically creates `Certificate` resources based on these annotations (in the same namespaces as those of Ingress resources with annotations).

i The Certificate for a particular annotation is linked to the existing Ingress resource. The additional records are put into the existing Ingress resource instead of creating a separate one. Thus, the process will fail if authentication or whitelist is set for the primary Ingress. In this case, you shouldn’t use the annotation; use the Certificate instead.

⚠ If you switched to the Certificate instead of annotation, delete the annotation-based Certificate. Otherwise, the same Secret will be updated for both Certificates, and this may lead to exceeding the Let’s Encrypt limits.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/tls-acme: "true"         # The annotation.
  name: example-com
  namespace: default
spec:
  ingressClassName: nginx
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: site
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
  - host: www.example.com                # The additional domain
    http:
      paths:
      - backend:
          service:
            name: site
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
  - host: admin.example.com               # Another additional domain
    http:
      paths:
      - backend:
          service:
            name: site
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
  tls:
  - hosts:
    - example.com
    - www.example.com                   # The additional domain
    - admin.example.com                  # Another additional domain
    secretName: example-com-tls           # The name of the certificate and secret
```

How do I check the certificate status?

```
d8 k -n default describe certificate example-com
...
Status:
  Acme:
    Authorizations:
      Account:  https://acme-v01.api.letsencrypt.org/acme/reg/22442061
      Domain:   example.com
      Uri:      https://acme-
v01.api.letsencrypt.org/acme/challenge/qJA9MGCZnUnVjAgxhoxONvDnKAsPatRILJ4n0lJ7MMY/4062050823
      Account:  https://acme-v01.api.letsencrypt.org/acme/reg/22442061
      Domain:   admin.example.com
      Uri:      https://acme-v01.api.letsencrypt.org/acme/challenge/pW2tFKLBDTll2Gx8UBqmEl846x5W-
YpBs8a4HqstJK8/4062050808
      Account:  https://acme-v01.api.letsencrypt.org/acme/reg/22442061
      Domain:   www.example.com
      Uri:      https://acme-v01.api.letsencrypt.org/acme/challenge/LaZJMM9_OKcTYbEThjT3oLtwgpkNfbHVdl8Dz-
yypx8/4062050792
  Conditions:
    Last Transition Time:  2018-04-02T18:01:04Z
    Message:               Certificate issued successfully
    Reason:                CertIssueSuccess
    Status:                True
    Type:                  Ready
Events:
  Type     Reason                 Age                From                     Message
  ----     ------                 ----               ----                     -------
  Normal   PrepareCertificate     1m                 cert-manager-controller  Preparing certificate with issuer
  Normal   PresentChallenge       1m                 cert-manager-controller  Presenting http-01 challenge for
domain example.com
  Normal   PresentChallenge       1m                 cert-manager-controller  Presenting http-01 challenge for
domain www.example.com
  Normal   PresentChallenge       1m                 cert-manager-controller  Presenting http-01 challenge for
domain admin.example.com
  Normal   SelfCheck              1m                 cert-manager-controller  Performing self-check for domain
admin.example.com
  Normal   SelfCheck              1m                 cert-manager-controller  Performing self-check for domain
example.com
  Normal   SelfCheck              1m                 cert-manager-controller  Performing self-check for domain
www.example.com
  Normal   ObtainAuthorization    55s                cert-manager-controller  Obtained authorization for domain
example.com
  Normal   ObtainAuthorization    54s                cert-manager-controller  Obtained authorization for domain
admin.example.com
  Normal   ObtainAuthorization    53s                cert-manager-controller  Obtained authorization for domain
www.example.com
```

How do I get a list of certificates?

```
d8 k get certificate --all-namespaces
NAMESPACE          NAME                         AGE
default            example-com                  13m
```

The “CAA record does not match issuer” error

Suppose `cert-manager` gets the following error when trying to provide a certificate:

```
CAA record does not match issuer
```

In this case, you have to check the `CAA (Certificate Authority Authorization)` DNS record of the domain for which the certificate is intended. For Let’s Encrypt certificates, the domain must have the `issue "letsencrypt.org"` CAA record. You can read more about CAA in the Let’s Encrypt documentation.

The chrony module

> **i** The module lifecycle stage: General Availability

Provides time synchronization on all cluster nodes using the chrony utility.

How it works

The module starts `chrony` agents on all cluster nodes.

By default, the NTP server `pool.ntp.org` is used. The NTP server can be changed via the settings module. To view the NTP servers used, you can use the command:

```
d8 k exec -it -n d8-chrony ds/chrony-master -- chronyc -N sources
```

Ouput example:

```
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* pool.ntp.org.                 2  10   377   171   -502us[ -909us] +/- 5388us
^- pool.ntp.org.                 2  10   377   666  -5317us[-5698us] +/-  103ms
^+ pool.ntp.org.                 2  10   377   938   -201us[ -567us] +/- 5346us
^+ pool.ntp.org.                 2  10   377   843   -159us[ -530us] +/-   12ms
```

> i
> `^+` - combined NTP server (`chrony` combines information from `combined` servers to reduce inaccuracies);
> `^*` - current NTP server;
> `^-` - non-combinable NTP server.
> Extra flag `-v` adds information about meanings of the columns: `chronyc sources -v`.

`chrony` agents on master nodes and on other nodes have one main difference - on all nodes that are not masters, the list of NTP servers contains not only NTP servers from `module config`, but also the addresses of all master nodes of the cluster.

Thus, agents on master nodes synchronize time only from the list of hosts specified in `module config` (by default from `pool.ntp.org`). And agents on other nodes synchronize time with the list of NTP servers from `module config` plus with `chrony` agents on master nodes.

This is done so that in case of unavailability of NTP servers specified in `module config`, the time is synchronized with the master nodes.

Third-party components

List of third-party software used in the `chrony` module:

- Chrony

  License: GNU General Public License version 2

  chrony is a versatile implementation of the Network Time Protocol (NTP)

Configuration

> i
> The module lifecycle stage: General Availability

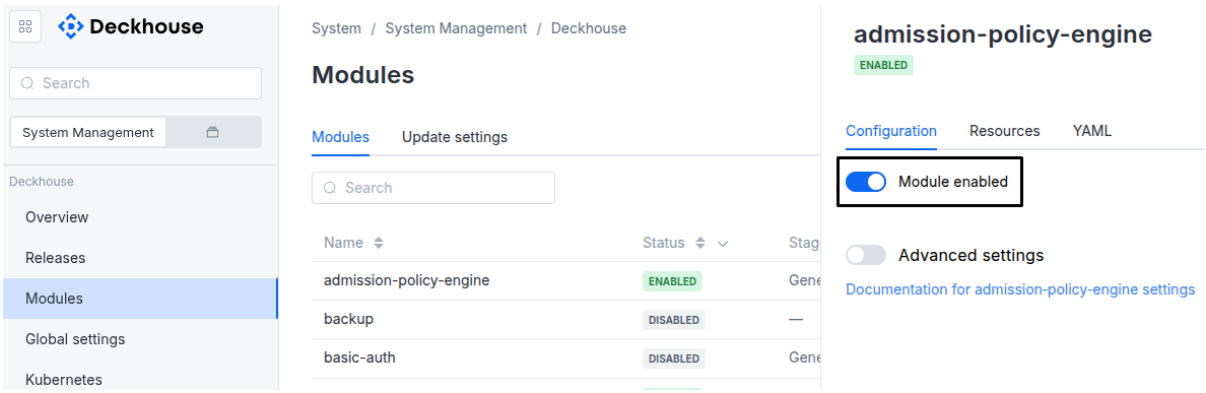
The module has 2 alerts.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `chrony` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable chrony
```

- **Using ModuleConfig `chrony`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `chrony` (create it if necessary);

  Example of a manifest to enable module `chrony`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: chrony
spec:
  enabled: true
```

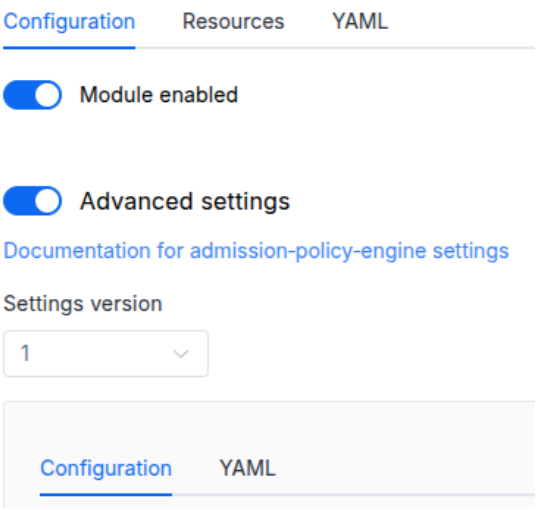
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `chrony` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `chrony` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `chrony`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `chrony` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc chrony
```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `chrony` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

  Example of a manifest for ModuleConfig `chrony`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: chrony
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.ntpServers**
    array of strings

    NTP servers to sync.

    Default: `["pool.ntp.org"]`

    Examples:

    ```
    ntpServers:
    - pool.ntp.org
    ```

    ```
    ntpServers:
    - pool.ntp.org
    - ntp.ubuntu.com
    - time.google.com
    ```

An example of the configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: chrony
spec:
  enabled: true
  settings:
    ntpServers:
      - pool.ntp.org
      - ntp.ubuntu.com
      - time.google.com
  version: 1
```

FAQ

> **i** The module lifecycle stage: General Availability

How do I disable chrony and use NTP daemon on nodes?

1. Disable usage of chrony module.
2. Create `NodeGroupConfiguration` custom step to enable use NTP daemon on nodes (example for `systemd-timesyncd`):

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: enable-ntp-on-node.sh
spec:
  weight: 100
  nodeGroups: ["*"]
  bundles: ["*"]
  content: |
    systemctl enable systemd-timesyncd
    systemctl start systemd-timesyncd
```

The cilium-hubble module

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

> **i** Web interfaces associated with the module: hubble

The `cilium-hubble` module provides visualization of the cluster network stack if the cilium CNI is enabled.

Requirements

The following is required for the `cilium-hubble` module:

- Linux kernel version >= 5.8 with eBPF support.
- BPF Type Format (BTF) support enabled. You can verify it as follows:
  - Run `ls -lah /sys/kernel/btf/vmlinux` — if the file exists, BTF is supported.
  - Run `grep -E "CONFIG_DEBUG_INFO_BTF=(y|m)" /boot/config-*` — if the parameter is enabled, BTF is supported.

Third-party components

List of third-party software used in the `cilium-hubble` module:

- Hubble UI 0.13.2

  License: Apache License 2.0

  Hubble UI is an open-source user interface for Cilium Hubble.

Configuration

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

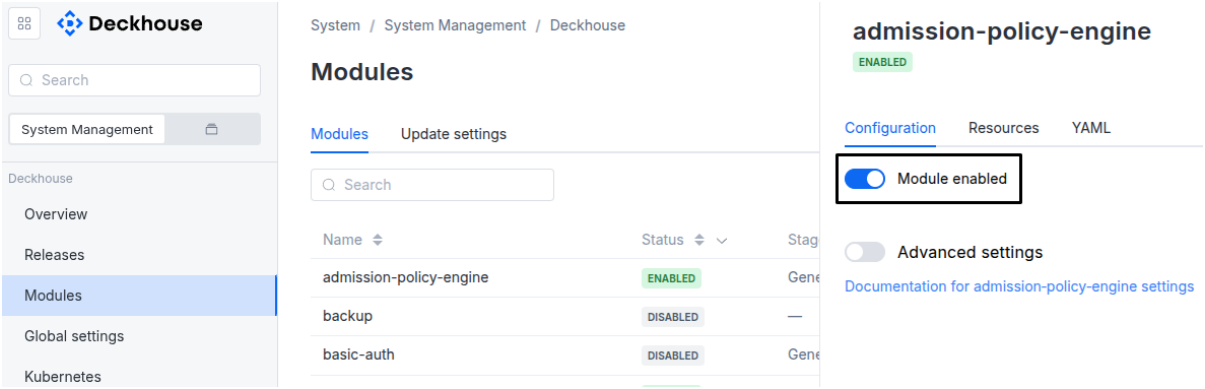
The module is not enabled by default in any bundles.

If the `cni-cilium` module is disabled, the `ciliumHubbleEnabled:` parameter will not affect the enabling of the `cilium-hubble` module.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cilium-hubble` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable cilium-hubble
  ```

- **Using ModuleConfig `cilium-hubble`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cilium-hubble` (create it if necessary);

  Example of a manifest to enable module `cilium-hubble`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cilium-hubble
  spec:
    enabled: true
  ```

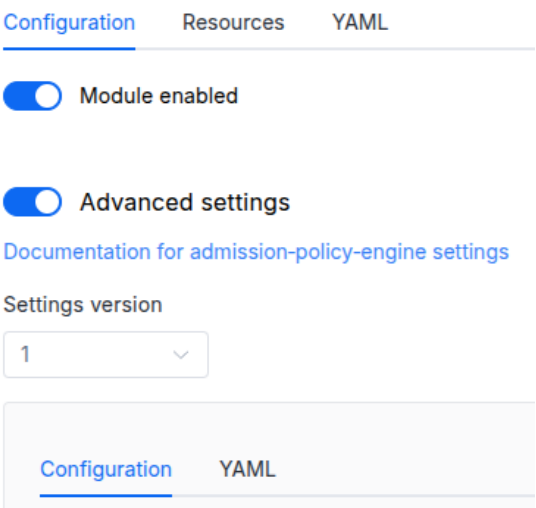
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cilium-hubble` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `cilium-hubble` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cilium-hubble`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

Edit the existing ModuleConfig `cilium-hubble` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc cilium-hubble
```

Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `cilium-hubble` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `cilium-hubble`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cilium-hubble
spec:
  version: 2
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Requirements

To the versions of other modules:

- `cni-cilium`: any version.

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Conversion description is missing.

  Actions performed by the conversion (jq syntax)...

  - `del(.auth.password) | if .auth == {} then del(.auth) end`

Parameters

Schema version: 2

- **settings**
  object
  - **settings.auth**
    object

    Options related to authentication or authorization in the Hubble web UI.

    - **settings.auth.allowedUserEmails**
      array of strings

      An array of emails of users that can access module’s public web interfaces.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

    - **settings.auth.allowedUserGroups**
      array of strings

      An array of user groups that can access Hubble web UI.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

      Caution! Note that you must add those groups to the appropriate field in the DexProvider config if this module is used together with the user-authn one.

- **settings.auth.externalAuthentication**
  object

  Parameters to enable external authentication based on the Ingress NGINX external-auth mechanism that uses the Nginx auth_request module.

  > **i** External authentication is enabled automatically if the user-authn module is enabled.

  - **settings.auth.externalAuthentication.authSignInURL**
    string

    The URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response code).

    Example:

    ```
    authSignInURL: https://example.com/dex/sign_in
    ```

  - **settings.auth.externalAuthentication.authURL**
    string

    The URL of the authentication service. If the user is authenticated, the service should return an HTTP 200 response code.

    Example:

    ```
    authURL: https://example.com/dex/auth
    ```

- **settings.auth.whitelistSourceRanges**
  array of strings

  An array if CIDRs that are allowed to authenticate in Hubble web UI.

  Example:

  ```
  whitelistSourceRanges:
  - 1.1.1.1/32
  ```

- **settings.debugLogging**
  boolean

  Enabled debug logging for Cilium Hubble component.

  Default: `false`

- **settings.https**
  object

  What certificate type to use.

  This parameter completely overrides the `global.modules.https` settings.

  Examples:

  ```
  https:
    mode: Disabled
  ```

  ```
  https:
    mode: OnlyInURI
  ```

  ```
  https:
    mode: CustomCertificate
    customCertificate:
      secretName: foobar
  ```

  ```
  https:
    mode: CertManager
    certManager:
      clusterIssuerName: letsencrypt
  ```

  - **settings.https.certManager**
    object

    Parameters for certmanager.

- **settings.https.certManager.clusterIssuerName**
  string

  What ClusterIssuer to use for getting an SSL certificate (currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available; also, you can define your own).

  Default: `letsencrypt`

  Examples:

  ```
  clusterIssuerName: letsencrypt
  ```

  ```
  clusterIssuerName: letsencrypt-staging
  ```

  ```
  clusterIssuerName: selfsigned
  ```

- **settings.https.customCertificate**
  object

  Parameters for custom certificate usage.

  - **settings.https.customCertificate.secretName**
    string

    The name of the secret in the `d8-system` namespace to use with the Hubble web UI.

    This secret must have the kubernetes.io/tls format.

- **settings.https.mode**
  string

  The HTTPS usage mode:

  - `CertManager` — the web UI is accessed over HTTPS using a certificate obtained from a clusterIssuer specified in the `certManager.clusterIssuerName` parameter;
  - `CustomCertificate` — the web UI is accessed over HTTPS using a certificate from the `d8-system` namespace;
  - `Disabled` — in this mode, the documentation web UI can only be accessed over HTTP;
  - `OnlyInURI` — the documentation web UI will work over HTTP (thinking that there is an external HTTPS load balancer in front of it that terminates HTTPS traffic). All the links in the `user-authn` will be generated using the HTTPS scheme. Load balancer should provide a redirect from HTTP to HTTPS.

  Default: `CertManager`

  Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller used for Hubble.

  Optional. By default, the `modules.ingressClass` global value is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.nodeSelector**
  object

  The same as the `spec.nodeSelector` pod parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

- **settings.tolerations**
  array of objects

  The same as `spec.tolerations` for the Kubernetes Pod.

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

Authentication

user-authn module provides authentication by default. Also, externalAuthentication can be configured. If these options are disabled, the module will use basic auth with the auto-generated password.

To view the generated password, run the command:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values cilium-hubble -o json | jq '.ciliumHubble.internal.auth.password'
```

To generate a new password, delete the Secret:

```
d8 k -n d8-cni-cilium delete secret/hubble-basic-auth
```

> **i** The `auth.password` parameter is deprecated.

Cloud provider — AWS

> **i** The module lifecycle stage: General Availability

The `cloud-provider-aws` module integrates Deckhouse Kubernetes Platform with Amazon AWS. It allows the `node-manager` module to use AWS resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-aws` module:

- Managing AWS resources via `cloud-controller-manager`:
  - creates network routes for the `PodNetwork` network on the AWS side;
  - creates load balancers for Services of the LoadBalancer type;
  - updates cluster node metadata and removes from Kubernetes nodes that no longer exist in AWS.
- Provisioning disks via the EBS CSI driver (`ebs.csi.aws.com`) and creating StorageClasses for AWS volume types so that PersistentVolumes can be requested from the cluster.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-aws`.
- Provisioning CloudEphemeral nodes via Machine Controller Manager (MCM). Virtual machine parameters are set in the AWSInstanceClass resource.
- Registering with `node-manager` so that AWSInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

Configuration

> **i** The module lifecycle stage: General Availability

The module is configured automatically based on the chosen placement strategy (the AWSClusterConfiguration custom resource). In most cases, you do not have to configure the module manually.

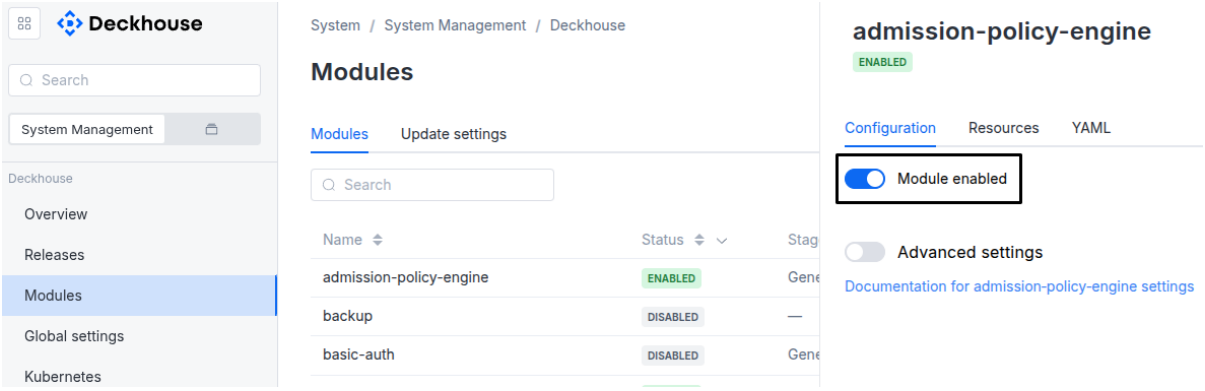
You can configure the number and parameters of provisioning machines in the cloud via the `NodeGroup` custom resource of the node-manager module. Also, in this custom resource, you can specify the instance class's name for the above group of nodes (the `cloudInstances.ClassReference` NodeGroup parameter). In the case of the AWS cloud provider, the instance class is the `AWSInstanceClass` custom resource that stores specific parameters of the machines.

The module automatically creates StorageClasses that are available in AWS: `gp3`, `gp2`, `sc1`, and `st1`. It lets you configure disks with the required IOPS. Also, it can filter out the unnecessary StorageClasses (you can do this via the `exclude` parameter).

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `cloud-provider-aws` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable cloud-provider-aws
  ```

- **Using ModuleConfig `cloud-provider-aws`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-aws` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-aws`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-aws
  spec:
    enabled: true
  ```

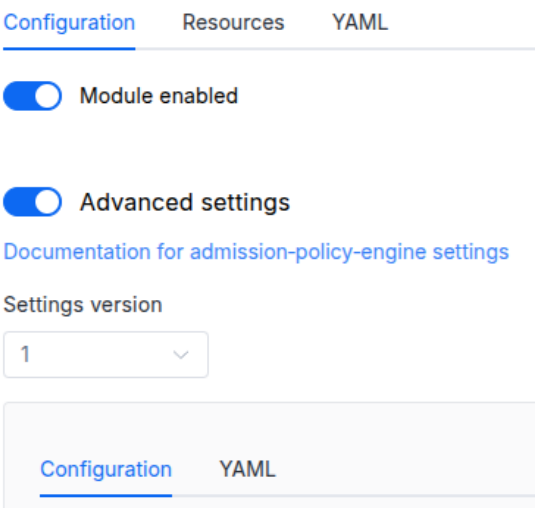
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-aws` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `cloud-provider-aws` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-aws`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

Edit the existing ModuleConfig `cloud-provider-aws` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc cloud-provider-aws
```

Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `cloud-provider-aws` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `cloud-provider-aws`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-aws
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.storageClass**
    object
    - **settings.storageClass.default**
      Deprecated
      string

      The name of StorageClass that will be used in the cluster by default.

      If the parameter is omitted, the default StorageClass is either:

      - an arbitrary StorageClass present in the cluster that has the default annotation;
      - the first (in lexicographic order) StorageClass created by the module.

      > **Parameter is deprecated.** Instead, use the global parameter global.defaultClusterStorageClass.

      Example:

      ```
      default: gp3
      ```

    - **settings.storageClass.exclude**
      array of strings

      A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

      Example:

      ```
      exclude:
      - sc.*
      - st1
      ```

    - **settings.storageClass.provision**
      array of objects

      Defines additional StorageClasses or override the parameters of created by default.

      - **settings.storageClass.provision.iops**
        string

I/O operations per second for `gp3` volume type:

- Allowed values: from `3000` to `16000`.
- If empty, it will set to 3000 as documented here.
- Changing doesn't apply to existing volumes. You can modify them manually through AWS Console.

- **settings.storageClass.provision.iopsPerGB**
string

  The number of I/O operations per second per GB for `io1` and `io2` volume types (this parameter is `3` for `gp2` volumes):

  - **Caution!** If the iopsPerGB value multiplied by the target volume's size is less than 100 or more than 64000, the creation of such a volume will fail.
  - You can find a detailed description of the volume types and their IOPS in the official documentation.
- **settings.storageClass.provision.name**
string

  Required value

  The name of the class to create.

- **settings.storageClass.provision.throughput**
string

  Throughput in MiB/s for `gp3` volume type:

  - Allowed values: from `125` to `1000`.
  - If empty, it will set to 125 as documented here.
  - Changing doesn't apply to existing volumes. You can modify them manually through AWS Console.
- **settings.storageClass.provision.type**
string

  Required value

  The volume type.

  Allowed values: `gp3`, `gp2`, `sc1`, `st1`, `io1`, `io2`

Provider configuration

> **i** The module lifecycle stage: General Availability

AWSClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in AWS.

Used by the cloud provider if a cluster's control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

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

> **i** After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithoutNAT
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 172.16.0.0/22
vpcNetworkCIDR: 172.16.0.0/16
masterNodeGroup:
  replicas: 1
  instanceClass:
    instanceType: m5.xlarge
    ami: ami-08b6d44b4f6f7b279
    diskType: gp3
nodeGroups:
- name: worker
  nodeTemplate:
    labels:
      node-role.kubernetes.io/worker: ''
  replicas: 2
  instanceClass:
    instanceType: t2.medium
    ami: ami-0aad10862ade98f27
  additionalTags:
    backup: srv1
provider:
  providerAccessKeyId: "<AWS_ACCESS_KEY>"
  providerSecretAccessKey: "<AWS_SECRET_ACCESS_KEY>"
  region: eu-central-1
tags:
  team: rangers
```

- **additionalRolePolicies**
  array of strings

  A list containing additional policy actions for IAM roles.

  - Additional policy actions would be attached to the default IAM role policy actions.
  - Parameter is optional. If omitted, only default IAM role policy actions are used.
  - Example of policy actions: `ecr:ListImages`, `s3:GetObject`, etc.

  Default IAM role policies actions contain the following roles:

  - `ec2:DescribeTags`
  - `ec2:DescribeInstances`
  - **Element of the array**
    string

    Pattern: `^([a-zA-Z0-9_\-]+):([a-zA-Z0-9_\-\*\?]+)$`

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **disableDefaultSecurityGroup**
  boolean

  If set to `true`, the default security group will not be created.

  Warning. When using `disableDefaultSecurityGroup: true`, you must manually create all required security groups to allow access to cluster nodes. Additionally, you must explicitly specify them in the following parameters:

  - `additionalSecurityGroups` in the `masterNodeGroup` section of the `AWSClusterConfiguration` resource;
  - `additionalSecurityGroups` in the `AWSInstanceClass` resource;
  - `additionalSecurityGroups` in the `nodeGroups.instanceClass` section.

  To configure the security groups used by load balancers, specify them using the `service.beta.kubernetes.io/aws-load-balancer-security-groups` annotation.

- **existingVPCID**
  string

  ID of the existing VPC to use for deploying.

  - **A mandatory parameter** if the `vpcNetworkCIDR` is omitted.
  - **Caution!** If there is an Internet Gateway in the target VPC, the deployment of the basic infrastructure will fail with an error. Currently, an Internet Gateway cannot be adopted.
- **iamNodeRole**
  string

  The name of the IAM role that will be attached to all AWS instances of the cluster nodes.

  DKP always creates and assigns a special IAM role named `<PREFIX>-node` to each AWS instance of the cluster nodes (`<PREFIX>` is the value of the parameter cloud.prefix in the common cluster parameters). You can specify your IAM-role with greater permissions in the `iamNodeRole` parameter, but it is important that it includes the IAM policies of the role created by DKP (the `<PREFIX>-node` role).

  More details about IAM roles for AWS EC2 can be found in the AWS documentation.

  Pattern: `^[a-zA-Z0-9+=,.@\-_']{1,64}$`
- **kind**
  string

  Required value

  Allowed values: `AWSClusterConfiguration`
- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

  > **i** **Note,** that the `Standard` layout is deprecated.

  Allowed values: `WithoutNAT`, `WithNAT`, `Standard`
- **masterNodeGroup**
  object

  Required value

  Parameters of the master's NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.additionalTags**
    object

    The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

    Example:

    ```
    additionalTags:
      project: cms-production
      severity: critical
    ```
  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the fields of the AWSInstanceClass.

    - **masterNodeGroup.instanceClass.additionalSecurityGroups**
      array of strings

      The additional security groups to add to provisioned instances of the specific InstanceClass.
    - **masterNodeGroup.instanceClass.ami**
      string

      Required value

      The Amazon Machine Image (AMI ID) to use in provisioned instances.

      Here is how you can find the required AMI (each region has its own set of AMIs):

```
aws ec2 --region <REGION> describe-images \
--filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-2020*' | \
jq '.Images[].ImageId'
```

Example:

```
ami: ami-040a1551f9c9d11ad
```

- **masterNodeGroup.instanceClass.diskSizeGb**
integer

  Instance disk size in gibibytes.

  Example:

  ```
  diskSizeGb: 50
  ```

- **masterNodeGroup.instanceClass.diskType**
string

  Instance EBS disk type.

  Allowed values: `gp3`, `gp2`, `io2`, `io1`, `st1`, `sc1`

  Example:

  ```
  diskType: gp2
  ```

- **masterNodeGroup.instanceClass.etcdDisk**
object

  Default: `{"sizeGb":20,"type":"gp3"}`

  - **masterNodeGroup.instanceClass.etcdDisk.sizeGb**
  integer

    Etcd disk size in gibibytes.

  - **masterNodeGroup.instanceClass.etcdDisk.type**
  string

    Etcd disk type.

- **masterNodeGroup.instanceClass.instanceType**
string

  Required value

  Instance type of AWS instance.

  Caution! Ensure that this type is present in all zones specified in the `zones` parameter.

  Example:

  ```
  instanceType: t3.large
  ```

- **masterNodeGroup.replicas**
integer

  Required value

  The number of master nodes to create.

  It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.zones**
array of strings

  Not required value.

  A limited set of zones in which master nodes can be created.

  The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.additionalTags**
    object

    The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

    Example:

    ```
    project: cms-production
    severity: critical
    ```

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the AWSInstanceClass.

    - **nodeGroups.instanceClass.additionalSecurityGroups**
      array of strings

      The additional security groups to add to provisioned instances of the specific InstanceClass.

    - **nodeGroups.instanceClass.ami**
      string

      Required value

      The Amazon Machine Image (AMI ID) to use in provisioned instances.

      Here is how you can find the required AMI (each region has its own set of AMIs):

      ```
      aws ec2 --region <REGION> describe-images \
      --filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-
      server-2020*' | \
      jq '.Images[].ImageId'
      ```

      Example:

      ```
      ami: ami-040a1551f9c9d11ad
      ```

    - **nodeGroups.instanceClass.diskSizeGb**
      integer

      Instance disk size in gibibytes.

      Example:

      ```
      diskSizeGb: 50
      ```

    - **nodeGroups.instanceClass.diskType**
      string

      Instance EBS disk type.

      Allowed values: `gp3`, `gp2`, `io2`, `io1`, `st1`, `sc1`

      Example:

      ```
      diskType: gp2
      ```

    - **nodeGroups.instanceClass.instanceType**
      string

      Required value

      Instance type of AWS instance.

      Caution! Ensure that this type is present in all zones specified in the `zones` parameter.

      Example:

      ```
      instanceType: t3.large
      ```

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup. It is used to generate the node name.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: "true"
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to cluster resources.

    The same as the `metadata.labels` standard field.

    Note that you have to re-create all the machines to add new tags if tags were modified in the running cluster.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object.

    > **i** Available fields: `effect`, `key`, and `values`.

    Example:

    ```
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
    ```

    - **nodeGroups.nodeTemplate.taints.effect**
      string

      Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

    - **nodeGroups.nodeTemplate.taints.key**
      string
    - **nodeGroups.nodeTemplate.taints.value**
      string

- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes.

- **nodeGroups.zones**
  array of strings

  Not required value.

  A limited set of zones in which nodes can be created.

  The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

- **nodeNetworkCIDR**
  string

  A subnet to use for cluster nodes:

  - The IP range must overlap or match the VPC address range.
  - The IP range will be evenly split into subnets, one per Availability Zone in your region.
  - An optional but recommended parameter. By default, it corresponds to the whole range of VPC addresses.

  If a new VPC is created along with a new cluster and no `vpcNetworkCIDR` is provided, then the range from `nodeNetworkCIDR` is used for the VPC. Thus, the entire VPC is allocated for the cluster networks, and you will not be able to add other resources to this VPC.

  The `nodeNetworkCIDR` range is distributed between subnets depending on the number of availability zones in the selected region. For example, if `nodeNetworkCIDR: "10.241.1.0/20"` and there are three availability zones in the region, subnets will be created with the `/22` mask.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **peeredVPCs**
  array of strings

  A list of AWS VPC IDs to peer with the cluster network.

  The service account must have access to all the VPCs listed. You have to configure the peering connection manually if no access is available.

- **provider**
  object

  Required value

  Contains settings to connect to the AWS API.

  - **provider.providerAccessKeyId**
    string

    Required value

    Access key ID.

  - **provider.providerSecretAccessKey**
    string

    Required value

    Access key secret.

  - **provider.region**
    string

    Required value

    The name of the AWS region where instances will be provisioned.

- **publicNetworkAllowList**
  array of strings

  A list of CIDRs allowed to access public endpoints of the cluster.

  Caution. AWS limits the number of inbound and outbound rules per security group to 60 each. If you need to specify more CIDRs, request a quota increase via AWS Support.

  Default: `["0.0.0.0/0"]`

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **sshAllowList**
  array of strings

  A list of CIDR’s allowed to connect to nodes via SSH.

  By default, `0.0.0.0/0`.

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **standard**
  Deprecated
  object

  Layout is **deprecated**.

- **tags**
  object

  Not required value.

  A dictionary of tags to create on all resources that support this feature.

  You have to re-create all the machines to add new tags if tags were modified in the running cluster.

- **vpcNetworkCIDR**
  string

  A subnet to use in the VPC being created.

  A mandatory parameter if the `existingVPCID` parameter is omitted.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **withNAT**
  object
  - **withNAT.bastionInstance**
    object
    - **withNAT.bastionInstance.instanceClass**
      object

      Required value

      Partial contents of the fields of the AWSInstanceClass.

      - **withNAT.bastionInstance.instanceClass.additionalSecurityGroups**
        array of strings

        The additional security groups to add to provisioned instance of the specific InstanceClass.

      - **withNAT.bastionInstance.instanceClass.ami**
        string

        Required value

        The Amazon Machine Image (AMI ID) to use in provisioned instance.

        Here is how you can find the required AMI (each region has its own set of AMIs):

        ```
        aws ec2 --region <REGION> describe-images \
        --filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-2020*' | \
        jq '.Images[].ImageId'
        ```

        Example:

        ```
        ami: ami-040a1551f9c9d11ad
        ```

      - **withNAT.bastionInstance.instanceClass.diskSizeGb**
        integer

        Instance disk size in gibibytes.

        Example:

        ```
        diskSizeGb: 20
        ```

      - **withNAT.bastionInstance.instanceClass.diskType**
        string

        Instance EBS disk type.

        Allowed values: `gp3`, `gp2`, `io2`, `io1`, `st1`, `sc1`

        Example:

        ```
        diskType: gp2
        ```

      - **withNAT.bastionInstance.instanceClass.instanceType**
        string

Required value

Instance type of AWS instance.

Caution! Ensure that this type is present in selected zone.

Example:

```
instanceType: t3.large
```

- **withNAT.bastionInstance.zone**
  string

  The zone in which the bastion instance will be created.

  By default, the first available zone in the region or the first from the list of the global parameter `zones` will be used.

- **withoutNAT**
  object
- **zones**
  array of strings

  The globally restricted set of zones that this cloud provider works with.

  Restricts only the **creation** of nodes in the cluster.

Preparing environment

> **i** The module lifecycle stage: General Availability

> ⚠ To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:
>
> - Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
> - `CgroupsV2` support.
> - Systemd version `244` or newer.
> - `erofs` kernel module support.
>
> For more information, see the documentation.
>
> In Amazon AWS, Ubuntu distributions with a custom kernel without `erofs` are used. To bring the OS into compliance with the requirements and add the necessary modules, you can use the command `sudo apt install linux-modules-extra-aws`.

> ⚠ The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:
>
> - `cloud-config.service`;
> - `cloud-final.service`;
> - `cloud-init.service`.

To use the `cloud-provider` and `machine-controller-manager` modules, you must access the AWS API as an IAM user with a sufficient set of privileges.

> ⚠ Make sure that you have access to the desired regions and that you have the necessary quotas.

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

JSON Policy

First, prepare a JSON file with the configuration of the necessary privileges:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeLaunchConfigurations",
                "autoscaling:DescribeTags",
                "ec2:AllocateAddress",
                "ec2:AssociateAddress",
                "ec2:AssociateRouteTable",
                "ec2:AttachInternetGateway",
                "ec2:AttachVolume",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CreateInternetGateway",
                "ec2:CreateKeyPair",
                "ec2:CreateNATGateway",
                "ec2:CreateRoute",
                "ec2:CreateRouteTable",
                "ec2:CreateSecurityGroup",
                "ec2:CreateSubnet",
                "ec2:CreateTags",
                "ec2:CreateVolume",
                "ec2:CreateVpc",
                "ec2:DeleteInternetGateway",
                "ec2:DeleteKeyPair",
                "ec2:DeleteNATGateway",
                "ec2:DeleteRoute",
                "ec2:DeleteRouteTable",
                "ec2:DeleteSecurityGroup",
                "ec2:DeleteSubnet",
                "ec2:DeleteTags",
                "ec2:DeleteVolume",
                "ec2:DeleteVpc",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAddresses",
                "ec2:DescribeAddressesAttribute",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeImages",
                "ec2:DescribeInstanceAttribute",
                "ec2:DescribeInstanceCreditSpecifications",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceTypes",
                "ec2:DescribeInstanceTopology",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeNatGateways",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeRegions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSecurityGroupRules",
                "ec2:DescribeSubnets",
                "ec2:DescribeTags",
                "ec2:DescribeVolumesModifications",
                "ec2:DescribeVolumes",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcClassicLink",
                "ec2:DescribeVpcClassicLinkDnsSupport",
                "ec2:DescribeVpcs",
                "ec2:DetachInternetGateway",
                "ec2:DetachVolume",
                "ec2:DisassociateAddress",
                "ec2:DisassociateRouteTable",
                "ec2:ImportKeyPair",
                "ec2:ModifyInstanceAttribute",
                "ec2:ModifySubnetAttribute",
                "ec2:ModifyVolume",
                "ec2:ModifyVpcAttribute",
```

```
                "ec2:ReleaseAddress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress",
                "ec2:RunInstances",
                "ec2:TerminateInstances",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:CreateVpcPeeringConnection",
                "ec2:DeleteVpcPeeringConnection",
                "ec2:AcceptVpcPeeringConnection",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "elasticloadbalancing:AddTags",
                "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
                "elasticloadbalancing:AttachLoadBalancerToSubnets",
                "elasticloadbalancing:ConfigureHealthCheck",
                "elasticloadbalancing:CreateListener",
                "elasticloadbalancing:CreateLoadBalancer",
                "elasticloadbalancing:CreateLoadBalancerListeners",
                "elasticloadbalancing:CreateLoadBalancerPolicy",
                "elasticloadbalancing:CreateTargetGroup",
                "elasticloadbalancing:DeleteListener",
                "elasticloadbalancing:DeleteLoadBalancer",
                "elasticloadbalancing:DeleteLoadBalancerListeners",
                "elasticloadbalancing:DeleteTargetGroup",
                "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
                "elasticloadbalancing:DeregisterTargets",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeLoadBalancerAttributes",
                "elasticloadbalancing:DescribeLoadBalancerPolicies",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeTargetHealth",
                "elasticloadbalancing:DetachLoadBalancerFromSubnets",
                "elasticloadbalancing:ModifyListener",
                "elasticloadbalancing:ModifyLoadBalancerAttributes",
                "elasticloadbalancing:ModifyTargetGroup",
                "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
                "elasticloadbalancing:RegisterTargets",
                "elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer",
                "elasticloadbalancing:SetLoadBalancerPoliciesOfListener",
                "iam:AddRoleToInstanceProfile",
                "iam:CreateInstanceProfile",
                "iam:CreateRole",
                "iam:CreateServiceLinkedRole",
                "iam:DeleteInstanceProfile",
                "iam:DeleteRole",
                "iam:DeleteRolePolicy",
                "iam:GetInstanceProfile",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListInstanceProfilesForRole",
                "iam:ListRolePolicies",
                "iam:ListAttachedRolePolicies",
                "iam:PassRole",
                "iam:PutRolePolicy",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:TagRole",
                "kms:DescribeKey",
                "sts:GetCallerIdentity"
            ],
            "Resource": "*"
        }
    ]
}
```

Below, you can find instructions on how to apply this policy.

Configuring IAM via the web interface

In order to configure IAM via the web interface, first create a new Policy and apply the previously created JSON file to it:

1. Open `Identity and Access Management (IAM)`.
2. Open the `Policies` page and click `Create Policy`.
3. Select the `JSON` tab and insert the policy.
4. Click `Next: Tags`, then `Next: Review`.
5. Enter a policy name in the `Name` field (e.g., `D8CloudProviderAWS`).
6. Click `Create Policy`.

Then add a new user:

1. Open the `Users` page of IAM and click `Add users`.
2. Enter a name in the `User name` field (e.g., `deckhouse`).

And apply the created Policy to it:

1. Select `Access key - Programmatic access` in the `Select AWS credential type` area.
2. Click `Next: Permissions`.
3. Select the `Attach existing policies directly` tab.
4. Search (use the `Filter policies` field) for the policy name entered above (e.g., `D8CloudProviderAWS`) and click the checkbox next to it.
5. Click `Next: Tags`, then `Next: Review`.
6. Click `Create user`.

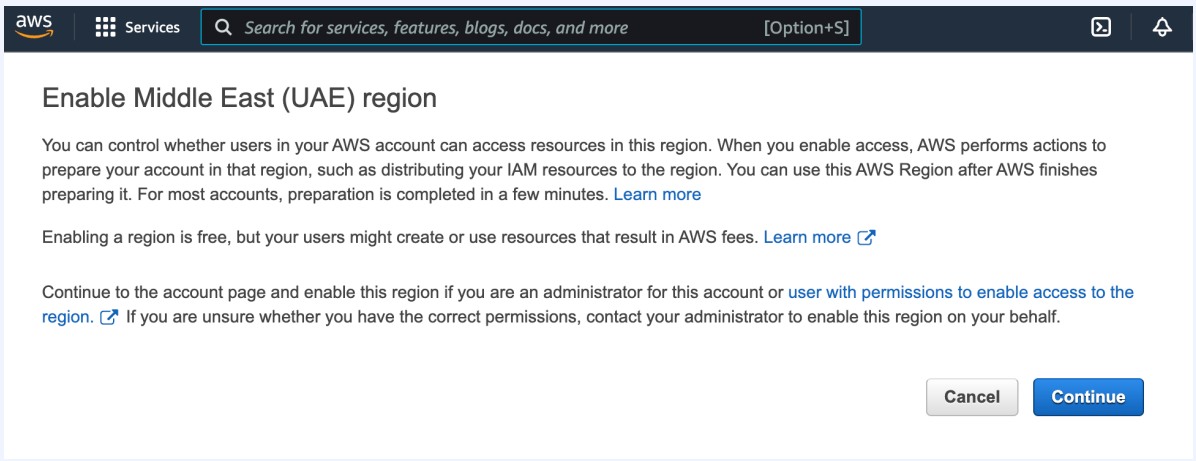
Save credentials (`Access key ID` and `Secret access key`).

i

Make sure that you have access to the desired regions. Select the desired region from the dropdown list in the upper right corner.

If there is a switch to the selected region, then there is access to the region.

If there is no access to the region, then you will receive the following message (may differ):



In this case, click `Continue` to enable the region.

Configuring IAM via the CLI

Create the `JSON specification` using the following command.

```
cat > policy.json << EOF
<Policy JSON spec>
EOF
```

Create a new Policy based on the specification created above with `D8CloudProviderAWS` as a policy name and the ARN identifier:

```
aws iam create-policy --policy-name D8CloudProviderAWS --policy-document file://policy.json
```

You will see the following:

```
{
   "Policy": {
      "PolicyName": "D8CloudProviderAWS",
      "PolicyId": "AAA",
      "Arn": "arn:aws:iam::123:policy/D8CloudProviderAWS",
      "Path": "/",
      "DefaultVersionId": "v1",
      "AttachmentCount": 0,
      "PermissionsBoundaryUsageCount": 0,
      "IsAttachable": true,
      "CreateDate": "2020-08-27T02:52:06+00:00",
      "UpdateDate": "2020-08-27T02:52:06+00:00"
   }
}
```

Create a new user:

```
aws iam create-user --user-name deckhouse
```

You will see the following:

```
{
   "User": {
      "Path": "/",
      "UserName": "deckhouse",
      "UserId": "AAAXXX",
      "Arn": "arn:aws:iam::123:user/deckhouse",
      "CreateDate": "2020-08-27T03:05:42+00:00"
   }
}
```

You need to allow access to the API and remember your `AccessKeyId` + `SecretAccessKey` values:

```
aws iam create-access-key --user-name deckhouse
```

You will see the following:

```
{
   "AccessKey": {
      "UserName": "deckhouse",
      "AccessKeyId": "XXXYYY",
      "Status": "Active",
      "SecretAccessKey": "ZZZzzz",
      "CreateDate": "2020-08-27T03:06:22+00:00"
   }
}
```

Attach the specified `Policy` to the specified `User`:

```
aws iam attach-user-policy --user-name username --policy-arn arn:aws:iam::123:policy/D8CloudProviderAWS
```

Configuring IAM via Terraform

An example of configuring IAM via Terraform:

```
resource "aws_iam_user" "user" {
  name = "deckhouse"
}

resource "aws_iam_access_key" "user" {
  user = aws_iam_user.user.name
}

resource "aws_iam_policy" "policy" {
  name       = "D8CloudProviderAWS"
  path       = "/"
  description = "Deckhouse policy"

  policy = <<EOF
<JSON Policy>
EOF
}

resource "aws_iam_user_policy_attachment" "policy-attachment" {
  user      = aws_iam_user.user.name
  policy_arn = aws_iam_policy.policy.arn
}
```

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

AWSInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of the AWS instance used by `machine-controller-manager` (the node-manager modile).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalSecurityGroups**
    array of strings

    The additional security groups to add to provisioned instances of the specific InstanceClass.

  - **spec.additionalTags**
    object

    Extra AWS tags for instances in addition to `AWSClusterConfiguration.tags`.

  - **spec.ami**
    string

    The Amazon Machine Image (AMI ID) to use in provisioned instances.

    If omitted, the AMI defined in the masterNodeGroup.instanceClass.ami parameter of the `AWSClusterConfiguration` resource is used.

    Here is how you can find the required AMI (each region has its own set of AMIs):

    ```
    aws ec2 --region <REGION> describe-images \
    --filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-2020*' | \
    jq '.Images[].ImageId'
    ```

Example:

```
ami: ami-040a1551f9c9d11ad
```

- **spec.capacity**
  Deprecated
  object

  Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

  Instance capacity for *non-standard* instance types (the `instanceTypes` parameter).

  Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

  The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`t2.micro`, `m6a.12xlarge`, etc.) is embedded in Deckhouse.

  - **spec.capacity.cpu**
    integer or string

    Required value

    Node vCPU resources.

    Pattern: `^[0-9]+m?$`

    Example:

    ```
    cpu: 1000m
    ```

  - **spec.capacity.memory**
    integer or string

    Required value

    Node memory resources.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    Example:

    ```
    memory: 1500Mi
    ```

- **spec.diskSizeGb**
  integer

  Instance root disk size in gibibytes.

  Default: `20`

  Example:

  ```
  diskSizeGb: 20
  ```

- **spec.diskType**
  string

  Instance EBS disk type.

  Default: `gp2`

  Example:

  ```
  diskType: gp2
  ```

- **spec.instanceType**
  string

  Required value

  Instance type of AWS instance.

  Caution! Ensure that this type is present in all zones specified in the `zones` parameter.

  Example:

  ```
  instanceType: t3.large
  ```

    - **spec.iops**
      string

      IOPS rate for io1 diskType

      Example:

      ```
      iops: '500'
      ```

    - **spec.spot**
      boolean

      Whether to create Spot Instances with minimum hourly price with which Instances will run.

- **status**
  object
    - **status.nodeGroupConsumers**
      array of strings

Parameters of the AWS instance used by `machine-controller-manager` (the node-manager modile).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

    - **spec.additionalSecurityGroups**
      array of strings

      The additional security groups to add to provisioned instances of the specific InstanceClass.

    - **spec.additionalTags**
      object

      Extra AWS tags for instances in addition to `AWSClusterConfiguration.tags`.

    - **spec.ami**
      string

      The Amazon Machine Image (AMI ID) to use in provisioned instances.

      If omitted, the AMI defined in the masterNodeGroup.instanceClass.ami parameter of the `AWSClusterConfiguration` resource is used.

      Here is how you can find the required AMI (each region has its own set of AMIs):

      ```
      aws ec2 --region <REGION> describe-images \
      --filters 'Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-2020*' | \
      jq '.Images[].ImageId'
      ```

      Example:

      ```
      ami: ami-040a1551f9c9d11ad
      ```

    - **spec.capacity**
      Deprecated
      object

      Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

      Instance capacity for *non-standard* instance types (the `instanceTypes` parameter).

      Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

      The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`t2.micro`, `m6a.12xlarge`, etc.) is embedded in Deckhouse.

        - **spec.capacity.cpu**
          integer or string

          Required value

          Node vCPU resources.

          Pattern: `^[0-9]+m?$`

Example:

```
cpu: 1000m
```

- **spec.capacity.memory**
  integer or string

  Required value

  Node memory resources.

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  Example:

```
memory: 1500Mi
```

- **spec.diskSizeGb**
  integer

  Instance root disk size in gibibytes.

  Default: `20`

  Example:

```
diskSizeGb: 20
```

- **spec.diskType**
  string

  Instance EBS disk type.

  Default: `gp2`

  Example:

```
diskType: gp2
```

- **spec.instanceType**
  string

  Required value

  Instance type of AWS instance.

  Caution! Ensure that this type is present in all zones specified in the `zones` parameter.

  Example:

```
instanceType: t3.large
```

- **spec.iops**
  string

  IOPS rate for io1 diskType

  Example:

```
iops: '500'
```

- **spec.spot**
  boolean

  Whether to create Spot Instances with minimum hourly price with which Instances will run.

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

FAQ

> **i** The module lifecycle stage: General Availability

How do I create a peering connection between VPCs?

Let’s, for example, create a peering connection between two VPCs, vpc-a and vpc-b.

> **Caution!** IPv4 CIDR must be unique for each VPC.

To configure, follow these steps:

1. Switch to the region where vpc-a is running.
2. Click `VPC` -> `VPC Peering Connections` -> `Create Peering Connection` and then configure a peering connection:
   - Name: `vpc-a-vpc-b`.
   - Fill in `Local` and `Another VPC` fields.
3. Switch to the region where vpc-b is running.
4. Click `VPC` -> `VPC Peering Connections`.
5. Select the newly created peering connection and click `Action "Accept Request"`.
6. Add routes to vpc-b’s CIDR over a peering connection to the vpc-a’s routing tables.
7. Add routes to vpc-a’s CIDR over a peering connection to the vpc-b’s routing tables.

How do I create a cluster in a new VPC with access over an existing bastion host?

1. Bootstrap the base-infrastructure of the cluster:

   ```
   dhctl bootstrap-phase base-infra --config config
   ```

2. Set up a peering connection using the instructions above.
3. Continue installing the cluster, enter `y` when asked about the Terraform cache:

   ```
   dhctl bootstrap --config config --ssh-...
   ```

How do I create a cluster in a new VPC and set up bastion host to access the nodes?

1. Bootstrap the base-infrastructure of the cluster:

   ```
   dhctl bootstrap-phase base-infra --config config
   ```

2. Manually set up the bastion host in the subnet -public-0.
3. Continue installing the cluster, enter `y` when asked about the Terraform cache:

   ```
   dhctl bootstrap --config config --ssh-...
   ```

Configuring a bastion host

There are two possible cases:

- A bastion host already exists in an external VPC; in this case, you need to:
  1. Create a basic infrastructure of the cluster: `dhctl bootstrap-phase base-infra`;
  2. Set up peering connection between an external and a newly created VPC;
  3. Continue the installation by specifying the bastion host: `dhctl bootstrap --ssh-bastion...`
- A bastion host needs to be deployed to a newly created VPC; in this case, you need to:
  1. Create a basic infrastructure of the cluster: `dhctl bootstrap-phase base-infra`;
  2. Manually run a bastion in the -public-0 subnet;
  3. Continue the installation by specifying the bastion host: `dhctl bootstrap --ssh-bastion...`

Adding CloudStatic nodes to a cluster

To add a pre-created VM as a node to a cluster, follow these steps:

1. Attach a security group `<prefix>-node` to the virtual machine.
2. Attach the IAM role `<prefix>-node` to the virtual machine.
3. Add the following tags to the virtual machine (so that `cloud-controller-manager` can find virtual machines in the cloud):

   ```
   "kubernetes.io/cluster/<cluster_uuid>" = "shared"
   "kubernetes.io/cluster/<prefix>" = "shared"
   ```

   - You can find out the `cluster_uuid` using the command:

     ```
     d8 k -n kube-system get cm d8-cluster-uuid -o json | jq -r '.data."cluster-uuid"'
     ```

- You can find out `prefix` using the command:

```
d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r '.data."cluster-
configuration.yaml"' \
  | base64 -d | grep prefix
```

How to increase the size of a volume?

Set the new size in the corresponding PersistentVolumeClaim resource, in the `spec.resources.requests.storage` parameter.

The operation is fully automatic and takes up to one minute. No further action is required.

The progress of the process can be observed in events using the command `d8 k describe pvc`.

> **i** After modifying a volume, you must wait at least six hours and ensure that the volume is in the `in-use` or `available` state before you can modify the same volume. This is sometimes referred to as a cooldown period. You can find details in the official documentation.

How do I configure access to Amazon ECR repository on cluster nodes?

1. Set permissions to read images in Repository policies. There must be an existing `Roles` object in `Principal`.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "RepositoryRead",
         "Effect": "Allow",
         "Principal": {
            "AWS": "arn:aws:iam::xxx:role/xxx-node"
         },
         "Action": [
           "ecr:BatchCheckLayerAvailability",
           "ecr:BatchGetImage",
           "ecr:DescribeImages",
           "ecr:DescribeRepositories",
           "ecr:GetAuthorizationToken",
           "ecr:GetDownloadUrlForLayer",
           "ecr:ListImages",
           "ecr:ListTagsForResource"
          ]
       }
     ]
   }
   ```

   Apply this policy in `Amazon ECR` > `Private registry` > `Repositories` > `` > `Permissions`.

2. Add `ecr:GetAuthorizationToken` to additionalRolePolicies.

What to do if switching to nodes in lower-priority groups takes a long time?

If switching to nodes in lower-priority groups takes a long time (for example, when node groups with spot instances are set to the highest priority and, if such instances are unavailable, switching to nodes from other groups takes a very long time), follow the instructions.

Examples

> **i** The module lifecycle stage: General Availability

An example of the `AWSInstanceClass` custom resource

Below is a simple example of custom resource `AWSInstanceClass` configuration:

```
apiVersion: deckhouse.io/v1
kind: AWSInstanceClass
metadata:
  name: worker
spec:
  instanceType: t3.large
  ami: ami-040a1551f9c9d11ad
  diskSizeGb: 15
  diskType:  gp2
```

LoadBalancer

Service object Annotations

The following parameters are supported in addition to the existing upstream ones:

1. `service.beta.kubernetes.io/aws-load-balancer-type` — if it has the `none` value, then the target group will **only** be created (without any LoadBalancer).
2. `service.beta.kubernetes.io/aws-load-balancer-backend-protocol` — this parameter is used together with `service.beta.kubernetes.io/aws-load-balancer-type: none`:
   - Possible values:
     - `tcp` (default);
     - `tls`;
     - `http`;
     - `https`.
   - **Caution!** The `cloud-controller-manager` (CCM) will try to recreate the target group in response to changes in this field. If the target group has NLB or ALB attached to it, the CCM will fail to delete it and get stuck in this state forever. You have to manually disconnect NLB or ALB from the target group.

Configuring security policies on nodes

There may be many reasons why you may need to restrict or expand incoming/outgoing traffic on cluster VMs in AWS:

- Allow VMs on a different subnet to connect to cluster nodes.
- Allow connecting to the ports of the static node so that the application can work.
- Restrict access to external resources or other VMs in the cloud for security reasons.

For all this, additional security groups should be used. You can only use security groups that are created in the cloud tentatively.

Enabling additional security groups on static and master nodes

This parameter can be set either in an existing cluster or when creating one. In both cases, additional security groups are declared in the `AWSClusterConfiguration`:

- for master nodes, in the `additionalSecurityGroups` field of the `masterNodeGroup` section;
- for static nodes, in the `additionalSecurityGroups` field of the `nodeGroups` subsection that corresponds to the target nodeGroup.

The `additionalSecurityGroups` field contains an array of strings with security group names.

Enabling additional security groups on ephemeral nodes

You have to set the `additionalSecurityGroups` parameter for all `AWSInstanceClass` that require additional security groups.

Configuring the load balancer if Ingress nodes are not available in all zones

Set the following annotation for the Service object: `service.beta.kubernetes.io/aws-load-balancer-subnets: subnet-foo, subnet-bar`.

You can get current subnets for a particular installation as follows:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values cloud-provider-aws -o json \
| jq -r '.cloudProviderAws.internal.zoneToSubnetIdMap'
```

Layouts

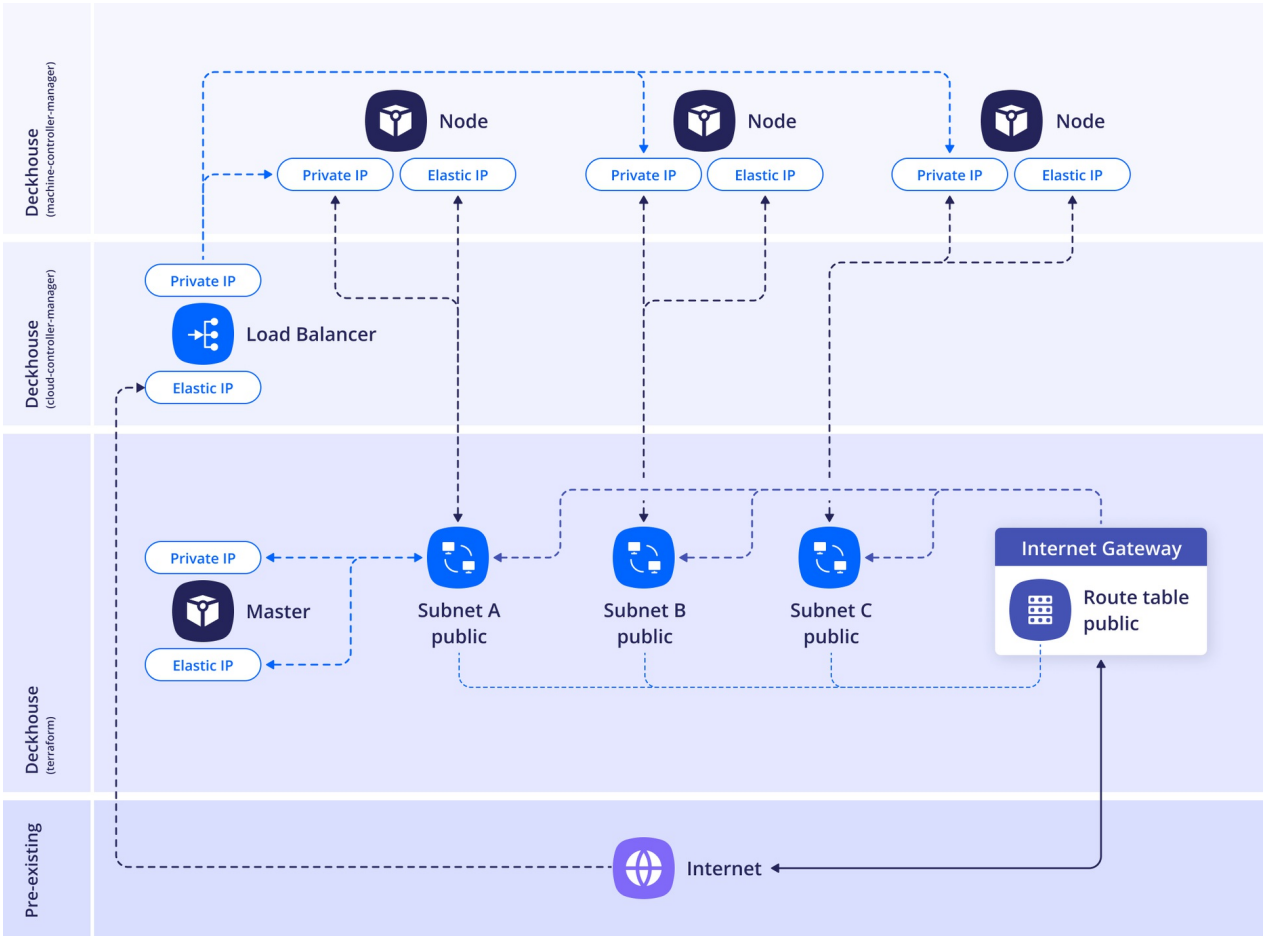
> **i** The module lifecycle stage: General Availability

Two layouts are supported. Below is more information about each of them.

WithoutNAT

Recommended layout.

Under this placement strategy, each node gets a public IP (ElasticIP). NAT is not used at all.



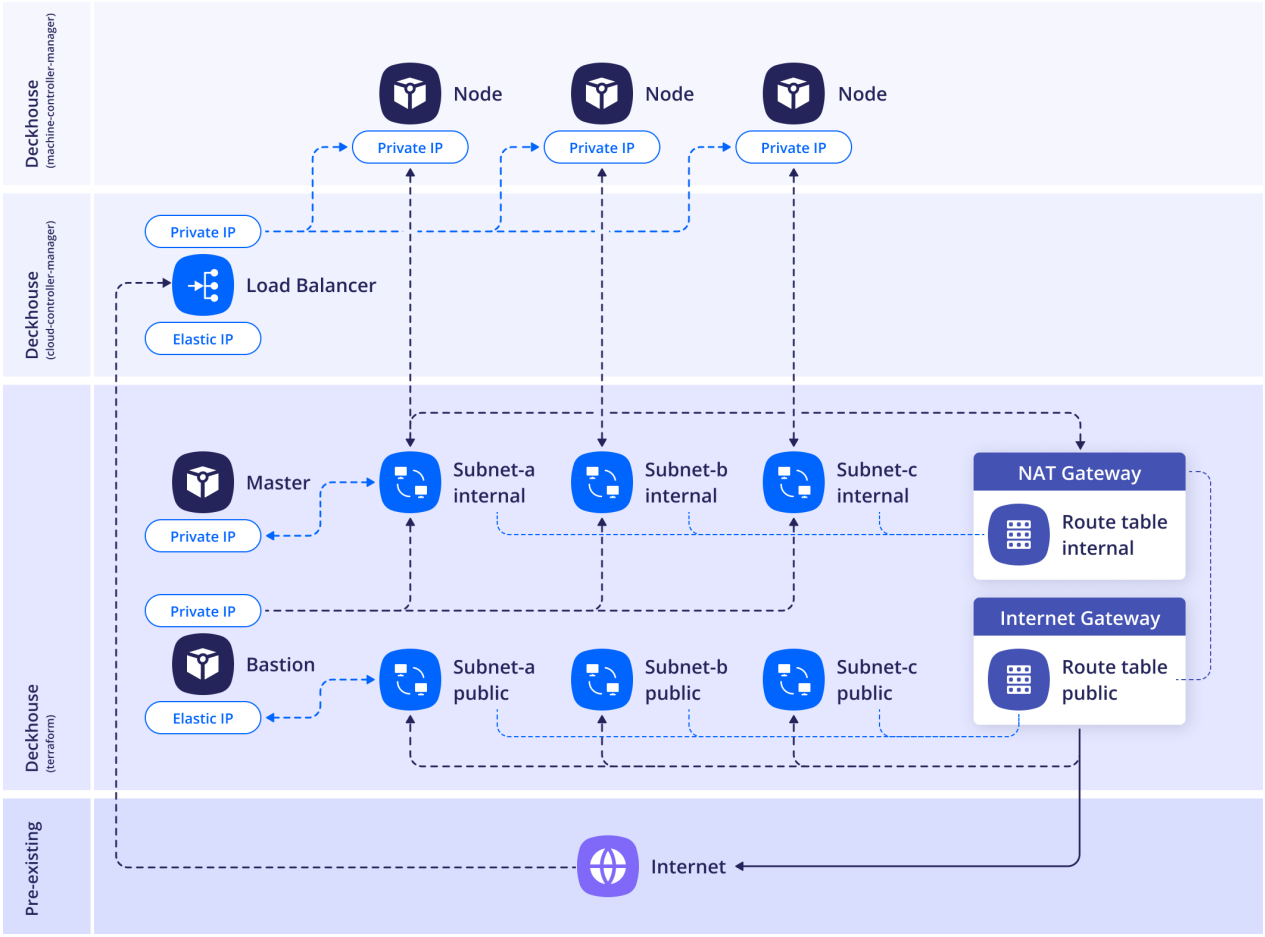
Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithoutNAT
vpcNetworkCIDR: "10.241.0.0/16"
nodeNetworkCIDR: "10.241.32.0/20"
sshPublicKey: <SSH_PUBLIC_KEY>
provider:
  providerAccessKeyId: '<AWS_ACCESS_KEY>'
  providerSecretAccessKey: '<AWS_SECRET_ACCESS_KEY>'
  region: eu-central-1
masterNodeGroup:
  replicas: 1
  instanceClass:
    # Type of the instance.
    instanceType: m5.xlarge
    # Amazon Machine Image ID
    # AMI Catalog in the AWS console: EC2 -> AMI Catalog
    ami: ami-0aad10862ade98f27
    # Master node VM disk size.
    diskSizeGb: 30
    # Master node VM disk type to use.
    diskType: gp3
nodeGroups:
  - name: mydb
    nodeTemplate:
      labels:
        node-role.kubernetes.io/mydb: ""
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aad10862ade98f27
    additionalTags:
      backup: srv1
tags:
  team: rangers
```

WithNAT

> **Caution!** A bastion host is required to access nodes (it can be created alongside the cluster by specifying the parameters in the section `withNAT.bastionInstance`).
>
> **Caution!** The NAT Gateway is always created in zone `a` in this layout. If cluster nodes are placed in other zones, then if there are problems in zone `a`, they will also be unavailable. In other words, when choosing the `WithNat` layout, the availability of the entire cluster will depend on the availability of zone `a`.

Virtual machines access the Internet using a NAT Gateway with a shared (and single) source IP.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: AWSClusterConfiguration
layout: WithNAT
provider:
  providerAccessKeyId: '<AWS_ACCESS_KEY>'
  providerSecretAccessKey: '<AWS_SECRET_ACCESS_KEY>'
  region: eu-central-1
withNAT:
  bastionInstance:
    zone: eu-central-1a
    instanceClass:
      instanceType: m5.large
      ami: ami-0aad10862ade98f27
      diskType: gp3
masterNodeGroup:
  # Number of master nodes.
  # If there is more than one master node, the etcd cluster will be set up automatically.
  replicas: 1
  instanceClass:
    # Type of the instance.
    instanceType: m5.xlarge
    # Amazon Machine Image ID.
    # AMI Catalog in the AWS console: EC2 -> AMI Catalog
    ami: ami-0aad10862ade98f27
    # Master node VM disk size.
    diskSizeGb: 30
    # Master node VM disk type to use.
    diskType: gp3
nodeGroups:
  - name: mydb
    nodeTemplate:
      labels:
        node-role.kubernetes.io/mydb: ""
    replicas: 2
    instanceClass:
      instanceType: t2.medium
      ami: ami-0aad10862ade98f27
    additionalTags:
      backup: me
vpcNetworkCIDR: "10.241.0.0/16"
nodeNetworkCIDR: "10.241.32.0/20"
sshPublicKey: <SSH_PUBLIC_KEY>
tags:
  team: rangers
```

Cloud provider — Azure

> **i** The module lifecycle stage: General Availability

The `cloud-provider-azure` module integrates Deckhouse Kubernetes Platform with Microsoft Azure. It allows the `node-manager` module to use Azure resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-azure` module:

- Managing Azure resources via `cloud-controller-manager`:
  - creates network routes for the `PodNetwork` network on the Azure side;
  - creates load balancers for Services of the LoadBalancer type;
  - updates cluster node metadata and removes from Kubernetes nodes that no longer exist in Azure.
- Provisioning disks via the Azure Disk CSI driver (`disk.csi.azure.com`) and creating StorageClasses for Azure disk types so that PersistentVolumes can be requested from the cluster.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-azurerm`.
- Provisioning CloudEphemeral nodes via Machine Controller Manager (MCM). Virtual machine parameters are set in the AzureInstanceClass resource.
- Registering with `node-manager` so that AzureInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

⚠ For utilities such as `ntpdate` and `chrony` to work correctly, make sure the load balancer has rules for UDP traffic. If outbound UDP is blocked, add a rule to the existing load balancer or create a Service of the LoadBalancer type with a UDP port.

Configuration

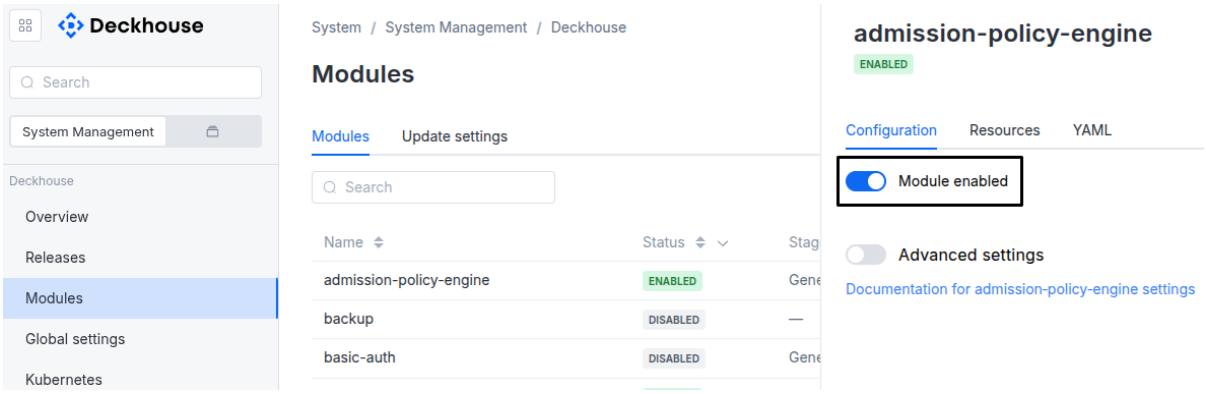
ℹ The module lifecycle stage: General Availability

The module is configured automatically based on the chosen placement strategy (the AzureClusterConfiguration custom resource). In most cases, you do not need to configure the module manually.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-azure` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cloud-provider-azure
```

- **Using ModuleConfig `cloud-provider-azure`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-azure` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-azure`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-azure
spec:
  enabled: true
```

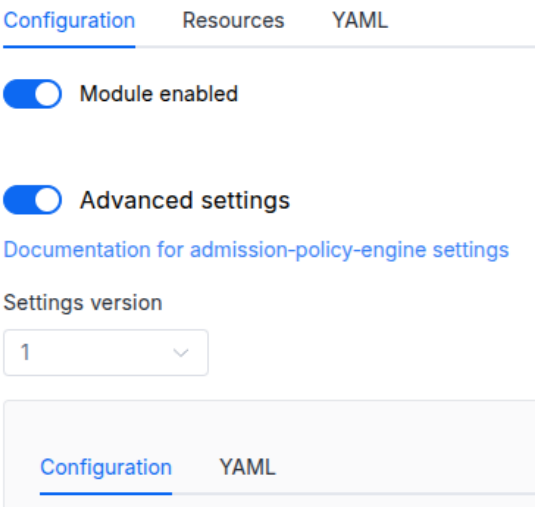
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-azure` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `cloud-provider-azure` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-azure`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-azure` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-azure
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cloud-provider-azure` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cloud-provider-azure`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-azure
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

You can configure the number and parameters of ordering machines in the cloud via the `NodeGroup` custom resource of the node-manager module. Also, in this custom resource, you can specify the instance class’s name for the above group of nodes (the cloudInstances.ClassReference parameter). In the case of the Azure cloud provider, the instance class is the `AzureInstanceClass` custom resource that stores specific parameters of the machines.

The module automatically creates the following StorageClasses:

Name	Disk type
managed-standard-ssd	StandardSSD_LRS
managed-standard	Standard_LRS
managed-premium	Premium_LRS

It allows you to configure additional StorageClasses for volumes with configurable IOPS and Throughput. Also, you can filter out the unnecessary StorageClasses via the exclude parameter.

An example of Storage Class configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-azure
spec:
  version: 1
  enabled: true
  settings:
    storageClass:
      provision:
      - name: managed-ultra-ssd
        diskIOPSReadWrite: 600
        diskMBpsReadWrite: 150
      exclude:
      - managed-standard.*
      - managed-premium
      default: managed-ultra-ssd
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.storageClass**
    object
    - **settings.storageClass.default**
      Deprecated
      string

      The name of StorageClass that will be used in the cluster by default.

      If the parameter is omitted, the default StorageClass is either:

      - existing StorageClass present in the cluster that has the default annotation;
      - the first (in lexicographic order) StorageClass created by the module.

      > **Parameter is deprecated.** Instead, use the global parameter global.defaultClusterStorageClass.

      Example:

      ```
      default: pd-ssd-not-replicated
      ```

    - **settings.storageClass.exclude**
      array of strings

      A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

      Example:

      ```
      exclude:
      - pd-standard.*
      - pd-ssd-replicated
      ```

    - **settings.storageClass.provision**
      array of objects

      Defines additional StorageClasses for Azure ultra disks.

      - **settings.storageClass.provision.cachingMode**
        string

        String value that corresponds to the desired caching mode. If expected disk size is more than 4 TiB, you have to set `cachineMode: None`.

        Check out Azure documentation for more information.

        Allowed values: `None`, `ReadOnly`, `ReadWrite`

      - **settings.storageClass.provision.diskIOPSReadWrite**
        number

Required value

Disk IOPS (limit of 300 IOPS/GiB, up to a maximum of 160 K IOPS per disk).

- **settings.storageClass.provision.diskMBpsReadWrite**
  number

  Required value

  Disk throughput in `MBps` (limit of a single disk is 256 KiB/s for each provisioned IOPS).

- **settings.storageClass.provision.name**
  string

  Required value

  The name of the class to create.

- **settings.storageClass.provision.tags**
  array of objects

  Azure disk tags.

  Example:

  ```
  tags:
  - key: key1
    value: val1
  - key: key2
    value: val2
  ```

  - **settings.storageClass.provision.tags.key**
    string

    Pattern: `^[A-Za-z0-9][-A-Za-z0-9_.]*$`

  - **settings.storageClass.provision.tags.value**
    string

    Pattern: `^[A-Za-z0-9][-A-Za-z0-9_.]*$`

- **settings.storageClass.provision.type**
  string

  Required value

  Azure disk storage account type.

  Check out Azure documentation for more information.

  Allowed values: `UltraSSD_LRS`

Provider configuration

> **i** The module lifecycle stage: General Availability

AzureClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in Azure.

Used by the cloud provider if a cluster’s control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

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

> **i** After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
vNetCIDR: 10.0.0.0/16
nameservers:
  addresses:
  - 8.8.8.8
  - 8.8.4.4
subnetCIDR: 10.0.0.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    machineSize: Standard_D4ds_v4
    urn: Canonical:UbuntuServer:18.04-LTS:18.04.202207120
    enableExternalIP: false
provider:
  subscriptionId: "<SUBSCRIPTION_ID>"
  location: westeurope
  clientId: "<CLIENT_ID>"
  clientSecret: "<CLIENT_SECRET>"
  tenantId: "<TENANT_ID>"
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`
- **kind**
  string

  Required value

  Allowed values: `AzureClusterConfiguration`
- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

  Allowed values: `Standard`
- **masterNodeGroup**

  Required value

  The definition of the master's NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the AzureInstanceClass fields.

    - **masterNodeGroup.instanceClass.acceleratedNetworking**
      boolean

      Accelerated Networking provides up to 30Gbps in networking throughput.

      Default: `false`
    - **masterNodeGroup.instanceClass.additionalTags**
      object

      The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

Example:

```
project: cms-production
severity: critical
```

- **masterNodeGroup.instanceClass.diskSizeGb**
  integer

  Instance root disk size in gibibytes.

  Example:

  ```
  diskSizeGb: 40
  ```

- **masterNodeGroup.instanceClass.diskType**
  string

  The type of the volumes to create.

  You can view a list of available volume types using the Azure CLI:

  ```
  az vm list-skus -l westeurope --zone
  ```

  Example:

  ```
  diskType: StandardSSD_LRS
  ```

- **masterNodeGroup.instanceClass.enableExternalIP**
  boolean

  Defines whether to enable external IP for an instance or not.

  Only available for the `Standard` layout.

  Default: `false`

  Allowed values: `true`, `false`

- **masterNodeGroup.instanceClass.etcdDiskSizeGb**
  integer

  Etcd disk size in gibibytes.

  Default: `20`

  Example:

  ```
  etcdDiskSizeGb: 20
  ```

- **masterNodeGroup.instanceClass.machineSize**
  string

  Required value

  The type of instances to provision.

  Getting a list of available types for the specific region using the Azure CLI:

  ```
  az vm list-sizes --location westeurope -o table
  ```

  Example:

  ```
  machineSize: Standard_F4
  ```

- **masterNodeGroup.instanceClass.urn**
  string

  Required value

  The VM image to use for an instance.

  More information about virtual machine images can be found in the official Azure documentation.

  Getting the `urn` via Azure CLI (column #4):

```
az vm image list --location westeurope --publisher Canonical --all --sku 20_04-lts -o
table
az vm image list --location westeurope --publisher OpenLogic --all --sku 7.5 -o table
```

By default, the image specified in `AzureCloudDiscoveryData` is used (the master of the cluster is based on this image).

The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version you use).

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create.

  It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.zones**
  array of strings

  A list of zones where master nodes can be created.

  You can browse a list of zones available for the selected instance type using the Azure CLI:

  ```
  az vm list-skus -l westeurope -o table
  ```

  Default: `[1,2,3]`

- **nameservers**
  object

  DNS server configuration for the Azure virtual network used by the cluster nodes.

  - **nameservers.addresses**
    array of strings

    A list of DNS servers.

    Example:

    ```
    addresses:
    - 8.8.8.8
    - 8.8.4.4
    ```

    - **Element of the array**
      string

      Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the AzureInstanceClass.

    - **nodeGroups.instanceClass.acceleratedNetworking**
      boolean

      Accelerated Networking provides up to 30Gbps in networking throughput.

      Default: `false`

    - **nodeGroups.instanceClass.additionalTags**
      object

      The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

      Example:

```
project: cms-production
severity: critical
```

- **nodeGroups.instanceClass.diskSizeGb**
  integer

  Instance root disk size in gibibytes.

  Example:

  ```
  diskSizeGb: 40
  ```

- **nodeGroups.instanceClass.diskType**
  string

  The type of the volumes to create.

  You can view a list of available volume types using the Azure CLI:

  ```
  az vm list-skus -l westeurope --zone
  ```

  Example:

  ```
  diskType: StandardSSD_LRS
  ```

- **nodeGroups.instanceClass.enableExternalIP**
  boolean

  Defines whether to enable external IP for an instance or not.

  Only available for the `Standard` layout.

  Default: `false`

  Allowed values: `true`, `false`

- **nodeGroups.instanceClass.machineSize**
  string

  Required value

  The type of instances to provision.

  Getting a list of available types for the specific region using the Azure CLI:

  ```
  az vm list-sizes --location westeurope -o table
  ```

  Example:

  ```
  machineSize: Standard_F4
  ```

- **nodeGroups.instanceClass.urn**
  string

  Required value

  The VM image to use for an instance.

  More information about virtual machine images can be found in the official Azure documentation.

  Getting the `urn` via Azure CLI (column #4):

  ```
  az vm image list --location westeurope --publisher Canonical --all --sku 20_04-lts -o
  table
  az vm image list --location westeurope --publisher OpenLogic --all --sku 7.5 -o table
  ```

  By default, the image specified in `AzureCloudDiscoveryData` is used (the master of the cluster is based on this image).

  The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version you use).

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: "true"
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to cluster resources.

    The same as the `metadata.labels` standard field.

    Note that you have to re-create all the machines to add new tags if tags were modified in the running cluster.

    Format — `key: value`.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object.

    > **i** Available fields: `effect`, `key`, and `values`.

    Example:

    ```
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
    ```

    - **nodeGroups.nodeTemplate.taints.effect**
      string

      Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

    - **nodeGroups.nodeTemplate.taints.key**
      string
    - **nodeGroups.nodeTemplate.taints.value**
      string

- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes to create.

- **nodeGroups.zones**
  array of strings

  A list of zones where static nodes can be created;

  You can browse a list of zones available for the selected instance type using the Azure CLI:

  ```
  az vm list-skus -l westeurope -o table
  ```

  Default: `[1,2,3]`

- **peeredVNets**
  array of objects

  An array of `VNets` to merge with the cluster network.

The service account must have access to all the `VNets` listed above. You have to configure the peering connection manually if no access is available.

- **peeredVNets.resourceGroupName**
  string

  Required value

  The name of the resource group with the VNet.
- **peeredVNets.vnetName**
  string

  Required value

  The name of the VNet.

- **provider**
  object

  Required value

  Parameters for connecting to the Azure API.

  - **provider.clientId**
    string

    Required value

    The client ID.
  - **provider.clientSecret**
    string

    Required value

    The client’s secret.

    Keep in mind the expiration date of the secret. By default, it is valid for one year. Refer to the official documentation to create a service account with a longer secret expiration date.
  - **provider.location**
    string

    Required value

    The name of the geo location to create all the resources. Getting available locations from Azure CLI:

    ```
    az account list-locations -o table
    ```
  - **provider.subscriptionId**
    string

    Required value

    The ID of the subscription.
  - **provider.tenantId**
    string

    Required value

    The ID of the tenant.

- **serviceEndpoints**
  array of strings

  The list of Service endpoints to associate with the subnet.

  Virtual Network (VNet) service endpoint provides secure and direct connectivity to Azure services over an optimized route over the Azure backbone network. Endpoints allow you to secure your critical Azure service resources to only your virtual networks. Service Endpoints enables private IP addresses in the VNet to reach the endpoint of an Azure service without needing a public IP address on the VNet.

  More information about Service Endpoints can be found in the official documentation.

  - **Element of the array**
    string

    Allowed values: `Microsoft.AzureActiveDirectory`, `Microsoft.AzureCosmosDB`, `Microsoft.ContainerRegistry`, `Microsoft.CognitiveServices`, `Microsoft.EventHub`, `Microsoft.KeyVault`, `Microsoft.ServiceBus`, `Microsoft.Sql`, `Microsoft.Storage`, `Microsoft.Storage.Global`, `Microsoft.Web`

- **sshAllowList**

array of strings

A list of CIDR’s allowed to connect to nodes via SSH.

By default, `*`.

- **sshPublicKey**
  string

  Required value

  Public key to access nodes as `azureuser`.

- **standard**
  object

  Settings for the `Standard` layout.

  - **standard.natGatewayPublicIpCount**
    integer

    The number of IP addresses for the NAT Gateway (pricing). `NAT Gateway` is not used if the value is `0`.

    Default: `0`

- **subnetCIDR**
  string

  Required value

  A subnet from the `vNetCIDR` address space for cluster nodes.

  A virtual network can be segmented into one or more subnets up to the limits.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  Example:

  ```
  subnetCIDR: 10.1.2.0/24
  ```

- **tags**
  object

  A list of tags in the `key: value` format to attach to all cluster resources.

  You have to re-create all the machines to add new tags if tags were modified in the running cluster.

- **vNetCIDR**
  string

  Required value

  An address space of the virtual network in the CIDR format.

  A virtual network is a virtual, isolated portion of the Azure public network. Each virtual network is dedicated to your subscription.

  Warning. If you are setting up peering, using vpn or linking networks of other clusters, network address spaces should not overlap.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  Example:

  ```
  vNetCIDR: 10.0.0.0/16
  ```

- **zones**
  array of strings

  The globally restricted set of zones that this Cloud Provider works with.

Preparing environment

> **i** The module lifecycle stage: General Availability

⚠ To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:

- Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- `CgroupsV2` support.
- Systemd version `244` or newer.
- `erofs` kernel module support.

For more information, see the documentation.

⚠ The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

⚠ **Caution!** Only regions where `Availability Zones` are available are supported.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

To rule the Microsoft Azure cloud, you need an account and at least a single Subscription connected to id.

You have to create a service account with Microsoft Azure so that Deckhouse Kubernetes Platform can manage cloud resources:

1. Install the Azure CLI, login and get Subscription ID:

   ```
   export SUBSCRIPTION_ID=$(az login | jq -r '.[0].id')
   ```

2. Create the service account:

   ```
   az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/$SUBSCRIPTION_ID" --
   name "DeckhouseCANDI"
   ```

   Example output of the command:

   ```
   {
     "appId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx",     <-- used in the clientId parameter
   of the AzureClusterConfiguration resource
     "displayName": "DeckhouseCANDI",
     "password": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", <-- used in the clientSecret
   parameter of the AzureClusterConfiguration resource
     "tenant": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"    <-- used in the tenantId parameter
   of the AzureClusterConfiguration resource
   }
   ```

   i By default, service account will be created with a secret (used in the clientSecret parameter of the `AzureClusterConfiguration` resource) validity period of one year without automatic renewal. Refer to the official documentation to create a service account with a longer secret expiration date.

You have to be logged in for further work with the `az` tool. Use the service account username, password, and tenant to log in:

```
az login --service-principal -u <username> -p <password> --tenant <tenant>
```

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

AzureInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of the Azure Instances used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.acceleratedNetworking**
    boolean

    Accelerated Networking provides up to 30Gbps in networking throughput.

    Default: `true`

  - **spec.additionalTags**
    object

    The additional tags to attach to the instances created.

  - **spec.capacity**
    Deprecated
    object

    Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

    Instance capacity for *non-standard* instance types (the `machineSize` parameter).

    Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

    The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`Standard_A1_v2`, `Standard_B16ms`, etc.) is embedded in Deckhouse.

    - **spec.capacity.cpu**
      integer or string

      Required value

      Node vCPU resources.

      Pattern: `^[0-9]+m?$`

      Example:

      ```
      cpu: 1000m
      ```

    - **spec.capacity.memory**
      integer or string

      Required value

      Node memory resources.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

      Example:

      ```
      memory: 1500Mi
      ```

  - **spec.diskSizeGb**
    integer

Instance root disk size in gibibytes.

Default: `50`

Example:

```
diskSizeGb: 40
```

- **spec.diskType**
  string

  The type of the volume to create.

  You can view a list of available volume types using the Azure CLI:

  ```
  az vm list-skus -l westeurope --zone`
  ```

  Default: `StandardSSD_LRS`

- **spec.machineSize**
  string

  Required value

  The type of instances to provision.

  You can view a list of available types for the specific region using the Azure CLI:

  ```
  az vm list-sizes --location westeurope -o table
  ```

  Example:

  ```
  machineSize: Standard_F4
  ```

- **spec.urn**
  string

  Not required value.

  The VM image to use for an instance.

  More information about virtual machine images can be found in the official documentation.

  You can get the latest version of an image using the Azure CLI:

  - `az vm image show --urn Canonical:UbuntuServer:18.04-LTS:latest --location westeurope`;
  - `az vm image show --urn Canonical:0001-com-ubuntu-server-focal:20_04-lts:latest --location westeurope`.

  By default, the image specified in `AzureCloudDiscoveryData` is used (the master of the cluster is based on this image).

  The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

  Example:

  ```
  urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Parameters of the Azure Instances used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.acceleratedNetworking**
    boolean

    Accelerated Networking provides up to 30Gbps in networking throughput.

    Default: `true`

- **spec.additionalTags**
  object

  The additional tags to attach to the instances created.

- **spec.capacity**
  Deprecated
  object

  Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

  Instance capacity for *non-standard* instance types (the `machineSize` parameter).

  Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

  The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`Standard_A1_v2`, `Standard_B16ms`, etc.) is embedded in Deckhouse.

  - **spec.capacity.cpu**
    integer or string

    Required value

    Node vCPU resources.

    Pattern: `^[0-9]+m?$`

    Example:

    ```
    cpu: 1000m
    ```

  - **spec.capacity.memory**
    integer or string

    Required value

    Node memory resources.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    Example:

    ```
    memory: 1500Mi
    ```

- **spec.diskSizeGb**
  integer

  Instance root disk size in gibibytes.

  Default: `50`

  Example:

  ```
  diskSizeGb: 40
  ```

- **spec.diskType**
  string

  The type of the volume to create.

  You can view a list of available volume types using the Azure CLI:

  ```
  az vm list-skus -l westeurope --zone`
  ```

  Default: `StandardSSD_LRS`

- **spec.machineSize**
  string

  Required value

  The type of instances to provision.

  You can view a list of available types for the specific region using the Azure CLI:

  ```
  az vm list-sizes --location westeurope -o table
  ```

Example:

```
machineSize: Standard_F4
```

- **spec.urn**
  string

  Not required value.

  The VM image to use for an instance.

  More information about virtual machine images can be found in the official documentation.

  You can get the latest version of an image using the Azure CLI:

  - `az vm image show --urn Canonical:UbuntuServer:18.04-LTS:latest --location westeurope`;
  - `az vm image show --urn Canonical:0001-com-ubuntu-server-focal:20_04-lts:latest --location westeurope`.

  By default, the image specified in `AzureCloudDiscoveryData` is used (the master of the cluster is based on this image).

  The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

  Example:

```
urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140
```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Examples

> **i** The module lifecycle stage: General Availability

An example of the `AzureInstanceClass` custom resource

Below is a simple example of the `AzureInstanceClass` custom resource:

```
apiVersion: deckhouse.io/v1
kind: AzureInstanceClass
metadata:
  name: example
spec:
  machineSize: Standard_F4
```

Layouts

> **i** The module lifecycle stage: General Availability

> **i** **Caution!** Only regions where `Availability Zones` are available are supported.

Standard

- A separate resource group is created for the cluster.
- By default, one external IP address is dynamically allocated to each instance (it is used for Internet access only). Each IP has 64000 ports available for SNAT.
- The NAT Gateway (pricing) is supported. With it, you can use static public IP addresses for SNAT.
- Public IP addresses can be assigned to master nodes and nodes created by Terraform.
- If the master does not have a public IP, then an additional instance with a public IP (aka bastion host) is required for installation tasks and access to the cluster. In this case, you will also need to configure peering between the cluster's VNet and bastion's VNet.
- Peering can also be configured between the cluster VNet and other VNets.

Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: AzureClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>" # Required.
vNetCIDR: 10.50.0.0/16 # Required.
subnetCIDR: 10.50.0.0/24 # Required.
standard:
  natGatewayPublicIpCount: 1 # Optional, by default 0.
masterNodeGroup:
  replicas: 1
  zones: ["1"] # Optional, by default ["1"].
  instanceClass:
    machineSize: Standard_F4 # Required.
    diskSizeGb: 32
    urn: Canonical:UbuntuServer:18.04-LTS:18.04.202010140 # Required.
    enableExternalIP: false # Optional, by default true.
provider:
  subscriptionId: "<SUBSCRIPTION_ID>" # Required.
  clientId: "<CLIENT_ID>" # Required.
  clientSecret: "<CLIENT_SECRET>" # Required.
  tenantId: "<TENANT_ID>" # Required.
  location: "westeurope" # Required.
# Optional, list of Azure VNets with which Kubernetes VNet will be peered.
peeredVNets:
  - resourceGroupName: kube-bastion # Required.
    vnetName: kube-bastion-vnet # Required.
```

Cloud provider — DVP

> **i** The module lifecycle stage: General Availability

The `cloud-provider-dvp` module integrates Deckhouse Kubernetes Platform with the Deckhouse Virtualization Platform. It allows the `node-manager` module to use DVP resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-dvp` module:

- Managing DVP resources via `cloud-controller-manager`: updates virtual machine and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in DVP.
- Provisioning disks via the DVP CSI driver (`csi.dvp.deckhouse.io`) so that PersistentVolumes can be requested from the cluster.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-kubernetes`.
- Provisioning CloudEphemeral nodes via Cluster API (CAPI). Virtual machine parameters are set in the DVPInstanceClass resource.
- Registering with `node-manager` so that DVPInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

> **⚠** If the cluster was installed with the DVPClusterConfiguration schema, migrate to configuration via ModuleConfig. Until the migration is complete, the `D8CloudProviderDVPMigrationPending` alert may fire and Deckhouse updates may be blocked.
>
> See How to migrate a cloud provider to ModuleConfig.

Configuration

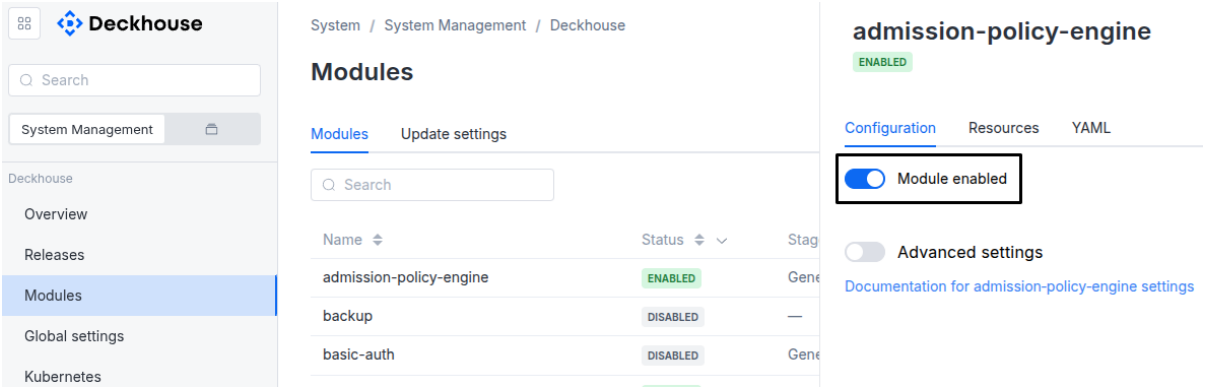
> **i** The module lifecycle stage: General Availability

The module is automatically enabled for all cloud clusters deployed in DVP.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-dvp` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable cloud-provider-dvp
  ```

- **Using ModuleConfig `cloud-provider-dvp`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-dvp` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-dvp`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-dvp
  spec:
    enabled: true
  ```

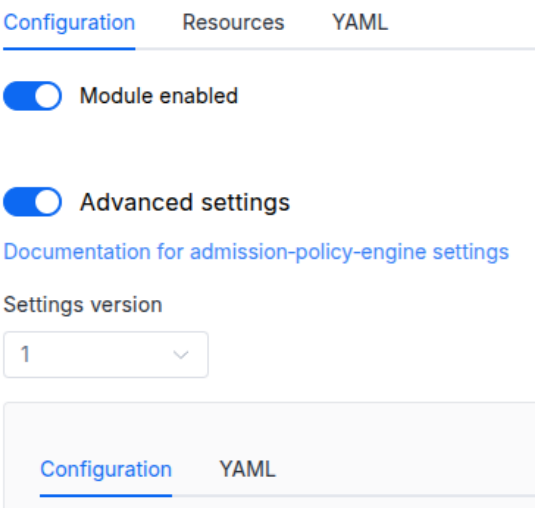
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-dvp` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `cloud-provider-dvp` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-dvp`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

Edit the existing ModuleConfig `cloud-provider-dvp` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc cloud-provider-dvp
```

Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `cloud-provider-dvp` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `cloud-provider-dvp`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-dvp
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.provider**
    object

    Contains settings to connect to the Deckhouse Kubernetes Platform API.

    - **settings.provider.kubeconfigDataBase64**
      string

      Required value

      Kubeconfig to connect to Deckhouse Kubernetes Platform API base64 encoded.

      Example:

      ```
      kubeconfigDataBase64: ZXhhbXBsZQo=
      ```

    - **settings.provider.namespace**
      string

      Required value

      Namespace in which DKP cluster resources will be created.

      > **i** Caution! If not explicitly specified, the default namespace for kubeconfig will be used.

      Example:

      ```
      namespace: default
      ```

  - **settings.zones**
    array of strings

    A set of zones in which nodes can be created. To use this setting, the `topology.kubernetes.io/zone` label must be set on DVP nodes. Read more about topological labels.

    > **i** Caution! To set the required label for a DVP node, follow the NodeGroup documentation.

    Example:

```
zones:
- zone-a
- zone-b
```

Provider configuration

> **i** The module lifecycle stage: General Availability

DVPClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in Deckhouse Virtualization Platform (DVP).

Used by the cloud provider if a cluster’s control plane is hosted in the DVP cloud.

Run the following command to change the configuration in a running cluster:

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

> **i** Once you modify the node configuration, run `dhctl converge` for changes to take effect.

Example:

```
apiVersion: deckhouse.io/v1
kind: DVPClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- zone-a
- zone-b
- zone-c
region: r1
masterNodeGroup:
  replicas: 3
  zones:
  - zone-a
  - zone-b
  - zone-c
  instanceClass:
    virtualMachine:
      cpu:
        cores: 1
        coreFraction: 100%
      memory:
        size: 4Gi
      virtualMachineClassName: generic
      ipAddresses:
      - 10.66.30.100
      - 10.66.30.101
      - 10.66.30.102
      additionalLabels:
        additional-vm-label: label-value
      additionalAnnotations:
        additional-vm-annotation: annotation-value
      tolerations:
      - key: dedicated.deckhouse.io
        operator: Equal
        value: system
      nodeSelector:
        beta.kubernetes.io/os: linux
    rootDisk:
      size: 10Gi
      storageClass: linstor-thin-r1
      image:
```

```
      kind: ClusterVirtualImage
      name: ubuntu-2204
   etcdDisk:
     size: 10Gi
     storageClass: linstor-thin-r1
   additionalDisks:
   - size: 10Gi
     storageClass: linstor-thin-r1
   - size: 1Gi
     storageClass: linstor-thin-r1
nodeGroups:
- name: worker
  zones:
  - zone-a
  - zone-b
  - zone-c
  replicas: 1
  instanceClass:
    virtualMachine:
      cpu:
        cores: 4
        coreFraction: 100%
      memory:
        size: 8Gi
      virtualMachineClassName: generic
    rootDisk:
      size: 10Gi
      image:
        kind: ClusterVirtualImage
        name: ubuntu-2204
    additionalDisks:
    - size: 10Gi
      storageClass: linstor-thin-r1
    - size: 1Gi
      storageClass: linstor-thin-r1
provider:
  kubeconfigDataBase64: ZXhhbXBsZQo=
  namespace: default
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`

- **kind**
  string

  Required value

  Allowed values: `DVPClusterConfiguration`

- **layout**
  string

  Required value

  Layout name.

  Read more about possible provider layouts.

  Allowed values: `Standard`

- **masterNodeGroup**
  object

  Required value

  The definition of the master’s NodeGroup.

  > **i** Once you modify the `masterNodeGroup` configuration, run [`dhctl converge`] for changes to take effect.

- **masterNodeGroup.instanceClass**
  object

  Required value

  Configuration for the virtual machine’s root disk.

  - **masterNodeGroup.instanceClass.additionalDisks**
    array of objects

    Parameters for additional virtual machine disks.

    Each item in the array describes one additional disk. Both `size` and `storageClass` parameters must be specified for each disk.

    - **masterNodeGroup.instanceClass.additionalDisks.size**
      string

      Required value

      Size of the additional disk.

      Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

      Example:

      ```
      size: 10Gi
      ```

    - **masterNodeGroup.instanceClass.additionalDisks.storageClass**
      string

      Required value

      Name of the existing StorageClass that will be used to create the additional disk.

      Example:

      ```
      storageClass: ceph-pool-r2-csi-rbd-immediate
      ```

  - **masterNodeGroup.instanceClass.additionalLabels**
    object

    Additional labels.

    Example:

    ```
    project: cms-production
    severity: critical
    ```

  - **masterNodeGroup.instanceClass.etcdDisk**
    object

    Required value

    Specifies settings for the etcd data disk.

    - **masterNodeGroup.instanceClass.etcdDisk.size**
      string

      Required value

      Etcd disk size.

      Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

      Example:

      ```
      size: 10Gi
      ```

    - **masterNodeGroup.instanceClass.etcdDisk.storageClass**
      string

      Name of the existing StorageClass will be used to create the etcd data disk.

      If the value is not specified, the StorageClass will be used according to the global storageClass parameter setting.

  - **masterNodeGroup.instanceClass.rootDisk**
    object

Required value

Specifies settings for the root disk of the virtual machine.

- **masterNodeGroup.instanceClass.rootDisk.image**
  object

  Required value

  Image parameters that will be used to create the virtual machine's root disk.

  - **masterNodeGroup.instanceClass.rootDisk.image.kind**
    string

    Required value

    The kind of the image source.

    Allowed values: `ClusterVirtualImage`, `VirtualImage`

  - **masterNodeGroup.instanceClass.rootDisk.image.name**
    string

    Required value

    The name of the image that will be used to create the root disk.

    > **i** The installation requires Linux OS images with cloud-init pre-installed.

- **masterNodeGroup.instanceClass.rootDisk.size**
  string

  Required value

  Root disk size.

  Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

  Example:

  ```
  size: 10Gi
  ```

- **masterNodeGroup.instanceClass.rootDisk.storageClass**
  string

  Name of the existing StorageClass will be used to create the virtual machine's root disk.

  If the value is not specified, the StorageClass will be used according to the global storageClass parameter setting.

- **masterNodeGroup.instanceClass.virtualMachine**
  object

  Required value

  Virtual machine settings for the created master node.

  > **i** The `runPolicy: AlwaysOnUnlessStoppedManually` policy is used for virtual machines of nodes. This allows the virtual machine to be stopped manually (for example, for maintenance) without triggering an automatic restart.

  - **masterNodeGroup.instanceClass.virtualMachine.additionalAnnotations**
    object

    Additional annotations for a virtual machine resource.

    Example:

    ```
    cluster-owner: user
    ```

  - **masterNodeGroup.instanceClass.virtualMachine.additionalLabels**
    object

    Additional labels for a virtual machine resource.

    Example:

```
cluster-owner: user
```

- **masterNodeGroup.instanceClass.virtualMachine.bootloader**
  string

  Defines a bootloader for the virtual machine.

  - `BIOS`: Use BIOS.
  - `EFI`: Use Unified Extensible Firmware (EFI/UEFI).
  - `EFIWithSecureBoot`: Use UEFI/EFI with Secure Boot support.

  Default: `EFI`

  Allowed values: `BIOS`, `EFI`, `EFIWithSecureBoot`

- **masterNodeGroup.instanceClass.virtualMachine.cpu**
  object

  Required value

  CPU settings for the virtual machine.

  - **masterNodeGroup.instanceClass.virtualMachine.cpu.coreFraction**
    string

    Guaranteed share of CPU that will be allocated to the virtual machine.

    Default: `100%`

    Pattern: `^100%$|^[1-9][0-9]?%$`

    Example:

    ```
    coreFraction: 100%
    ```

  - **masterNodeGroup.instanceClass.virtualMachine.cpu.cores**
    integer

    Required value

    Number of CPU cores for the virtual machine.

    Allowed values: `1 <= X`

- **masterNodeGroup.instanceClass.virtualMachine.ipAddresses**
  array of strings

  Static IP addresses to be assigned to the network interfaces of the virtual machines. The number of addresses must match the number of replicas being created — each IP address will be assigned to a specific virtual machine replica. For example, if 3 replicas are specified and the IP addresses provided are: ip1, ip2, and ip3, then ip1 will be assigned to the first replica, ip2 to the second, and ip3 to the third.

  > **i** These addresses must belong to the address range specified in the virtualization module configuration in the `virtualMachineCIDRs` parameter.

  - **Element of the array**
    string

    Pattern: `^([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})|(Auto)$`

- **masterNodeGroup.instanceClass.virtualMachine.liveMigrationPolicy**
  string

  Live migration policy for the virtual machine.

  - `Manual`: Migration is controlled manually.
  - `Never`: Migration is disabled.
  - `AlwaysSafe`: Always use safe migration (may fail if VM has a high rate of memory changes).
  - `PreferSafe`: Prefer safe migration, fallback to forced if needed.
  - `AlwaysForced`: Always use forced migration with VM slowdown.
  - `PreferForced`: Prefer forced migration (recommended for master nodes due to high memory activity).

  Default: `PreferForced`

  Allowed values: `Manual`, `Never`, `AlwaysSafe`, `PreferSafe`, `AlwaysForced`, `PreferForced`

- **masterNodeGroup.instanceClass.virtualMachine.memory**

object

Required value

Specifies the memory settings for the virtual machine.

- **masterNodeGroup.instanceClass.virtualMachine.memory.size**
  string

  Required value

  Amount of memory resources allowed for the virtual machine.

  Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

  Example:

  ```
  size: 4Gi
  ```

- **masterNodeGroup.instanceClass.virtualMachine.nodeSelector**
  object

  Allows a virtual machine to be assigned to specified DVP nodes. The same as in the `spec.nodeSelector` parameter for Kubernetes Pods.

- **masterNodeGroup.instanceClass.virtualMachine.priorityClassName**
  string

  The same as in the `spec.priorityClassName` parameter for Kubernetes Pods.

- **masterNodeGroup.instanceClass.virtualMachine.runPolicy**
  string

  Virtual machine run policy.

  - `AlwaysOn`: The virtual machine should always be running.
  - `AlwaysOff`: The virtual machine should always be stopped.
  - `Manual`: The virtual machine state is controlled manually.
  - `AlwaysOnUnlessStoppedManually`: The virtual machine can be stopped manually (for example, for maintenance), but it will automatically start after a host reboot.

  Default: `AlwaysOnUnlessStoppedManually`

  Allowed values: `AlwaysOn`, `AlwaysOff`, `Manual`, `AlwaysOnUnlessStoppedManually`

- **masterNodeGroup.instanceClass.virtualMachine.tolerations**
  array of objects

  Allows setting tolerations for virtual machines for a DVP node. The same as in the `spec.tolerations` parameter in Kubernetes Pods.

  - **masterNodeGroup.instanceClass.virtualMachine.tolerations.effect**
    string
  - **masterNodeGroup.instanceClass.virtualMachine.tolerations.key**
    string
  - **masterNodeGroup.instanceClass.virtualMachine.tolerations.operator**
    string
  - **masterNodeGroup.instanceClass.virtualMachine.tolerations.tolerationSeconds**
    integer
  - **masterNodeGroup.instanceClass.virtualMachine.tolerations.value**
    string

- **masterNodeGroup.instanceClass.virtualMachine.virtualMachineClassName**
  string

  Required value

  The name of the VirtualMachineClass.

  Intended for centralized configuration of preferred virtual machine parameters. It allows you to specify CPU instruction sets, resource configuration policies for CPU and memory, and define the ratio between these resources.

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create.

  It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.zones**
  array of strings

  A set of zones in which nodes can be created.

  To use this setting, the `topology.kubernetes.io/zone` label must be set on DVP nodes. Read more about topological labels.

  > **i** To set the required label for a DVP node, follow the NodeGroup documentation.

- **nodeGroups**
  array of objects

  Array of additional NodeGroups for creating static nodes (for example, for dedicated frontend nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Configuration of the virtual machine and its disk for the created static node.

    - **nodeGroups.instanceClass.additionalDisks**
      array of objects

      Parameters for additional virtual machine disks.

      Each item in the array describes one additional disk. Both `size` and `storageClass` parameters must be specified for each disk.

      - **nodeGroups.instanceClass.additionalDisks.size**
        string

        Required value

        Size of the additional disk.

        Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

        Example:

        ```
        size: 10Gi
        ```

      - **nodeGroups.instanceClass.additionalDisks.storageClass**
        string

        Required value

        Name of the existing StorageClass that will be used to create the additional disk.

        Example:

        ```
        storageClass: ceph-pool-r2-csi-rbd-immediate
        ```

    - **nodeGroups.instanceClass.rootDisk**
      object

      Required value

      Specifies settings for the root disk of the virtual machine.

      - **nodeGroups.instanceClass.rootDisk.image**
        object

        Required value

        Image parameters that will be used to create the virtual machine’s root disk.

        - **nodeGroups.instanceClass.rootDisk.image.kind**
          string

          Required value

          The kind of the image source.

          Allowed values: `ClusterVirtualImage`, `VirtualImage`

        - **nodeGroups.instanceClass.rootDisk.image.name**
          string

Required value

The name of the image that will be used to create the root disk.

> **i** The installation requires Linux OS images with cloud-init pre-installed.

- **nodeGroups.instanceClass.rootDisk.size**
  string

  Required value

  Root disk size.

  Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

  Example:

  ```
  size: 10Gi
  ```

- **nodeGroups.instanceClass.rootDisk.storageClass**
  string

  Name of the existing StorageClass will be used to create the virtual machine's root disk.

  If the value is not specified, the StorageClass will be used according to the global storageClass parameter setting.

- **nodeGroups.instanceClass.virtualMachine**
  object

  Required value

  Virtual machine settings for the created master node.

  > **i** The `runPolicy: AlwaysOnUnlessStoppedManually` policy is used for virtual machines of nodes. This allows the virtual machine to be stopped manually (for example, for maintenance) without triggering an automatic restart.

  - **nodeGroups.instanceClass.virtualMachine.additionalAnnotations**
    object

    Additional annotations for a virtual machine resource.

    Example:

    ```
    cluster-owner: user
    ```

  - **nodeGroups.instanceClass.virtualMachine.additionalLabels**
    object

    Additional labels for a virtual machine resource.

    Example:

    ```
    cluster-owner: user
    ```

  - **nodeGroups.instanceClass.virtualMachine.bootloader**
    string

    Defines a bootloader for the virtual machine.

    - `BIOS`: Use BIOS.
    - `EFI`: Use Unified Extensible Firmware (EFI/UEFI).
    - `EFIWithSecureBoot`: Use UEFI/EFI with Secure Boot support.

    Default: `EFI`

    Allowed values: `BIOS`, `EFI`, `EFIWithSecureBoot`

  - **nodeGroups.instanceClass.virtualMachine.cpu**
    object

    Required value

    CPU settings for the virtual machine.

    - **nodeGroups.instanceClass.virtualMachine.cpu.coreFraction**

string

Guaranteed share of CPU that will be allocated to the virtual machine.

Default: `100%`

Pattern: `^100%$|^[1-9][0-9]?%$`

Example:

```
coreFraction: 100%
```

- **nodeGroups.instanceClass.virtualMachine.cpu.cores**
  integer

  Required value

  Number of CPU cores for the virtual machine.

  Allowed values: `1 <= X`

- **nodeGroups.instanceClass.virtualMachine.ipAddresses**
  array of strings

  Static IP addresses to be assigned to the network interfaces of the virtual machines. The number of addresses must match the number of replicas being created — each IP address will be assigned to a specific virtual machine replica. For example, if 3 replicas are specified and the IP addresses provided are: ip1, ip2, and ip3, then ip1 will be assigned to the first replica, ip2 to the second, and ip3 to the third.

  > **i** These addresses must belong to the address range specified in the virtualization module configuration in the `virtualMachineCIDRs` parameter.

  - **Element of the array**
    string

    Pattern: `^([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})|(Auto)$`

- **nodeGroups.instanceClass.virtualMachine.liveMigrationPolicy**
  string

  Live migration policy for the virtual machine.

  - `Manual`: Migration is controlled manually.
  - `Never`: Migration is disabled.
  - `AlwaysSafe`: Always use safe migration (may fail if VM has a high rate of memory changes).
  - `PreferSafe`: Prefer safe migration, fallback to forced if needed.
  - `AlwaysForced`: Always use forced migration with VM slowdown.
  - `PreferForced`: Prefer forced migration (recommended for master nodes due to high memory activity).

  Default: `PreferForced`

  Allowed values: `Manual`, `Never`, `AlwaysSafe`, `PreferSafe`, `AlwaysForced`, `PreferForced`

- **nodeGroups.instanceClass.virtualMachine.memory**
  object

  Required value

  Specifies the memory settings for the virtual machine.

  - **nodeGroups.instanceClass.virtualMachine.memory.size**
    string

    Required value

    Amount of memory resources allowed for the virtual machine.

    Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

    Example:

    ```
    size: 4Gi
    ```

- **nodeGroups.instanceClass.virtualMachine.nodeSelector**
  object

Allows a virtual machine to be assigned to specified DVP nodes. The same as in the `spec.nodeSelector` parameter for Kubernetes Pods.

- **nodeGroups.instanceClass.virtualMachine.priorityClassName**
  string

  The same as in the `spec.priorityClassName` parameter for Kubernetes Pods.

- **nodeGroups.instanceClass.virtualMachine.runPolicy**
  string

  Virtual machine run policy.

  - `AlwaysOn`: The virtual machine should always be running.
  - `AlwaysOff`: The virtual machine should always be stopped.
  - `Manual`: The virtual machine state is controlled manually.
  - `AlwaysOnUnlessStoppedManually`: The virtual machine can be stopped manually (for example, for maintenance), but it will automatically start after a host reboot.

  Default: `AlwaysOnUnlessStoppedManually`

  Allowed values: `AlwaysOn`, `AlwaysOff`, `Manual`, `AlwaysOnUnlessStoppedManually`

- **nodeGroups.instanceClass.virtualMachine.tolerations**
  array of objects

  Allows setting tolerations for virtual machines for a DVP node. The same as in the `spec.tolerations` parameter in Kubernetes Pods.

  - **nodeGroups.instanceClass.virtualMachine.tolerations.effect**
    string
  - **nodeGroups.instanceClass.virtualMachine.tolerations.key**
    string
  - **nodeGroups.instanceClass.virtualMachine.tolerations.operator**
    string
  - **nodeGroups.instanceClass.virtualMachine.tolerations.tolerationSeconds**
    integer
  - **nodeGroups.instanceClass.virtualMachine.tolerations.value**
    string

- **nodeGroups.instanceClass.virtualMachine.virtualMachineClassName**
  string

  Required value

  The name of the VirtualMachineClass.

  Intended for centralized configuration of preferred virtual machine parameters. It allows you to specify CPU instruction sets, resource configuration policies for CPU and memory, and define the ratio between these resources.

- **nodeGroups.name**
  string

  Required value

  Name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the standard `metadata.annotations` field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: "true"
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to all cluster resources (if supported by the resources).

    The same as the standard `metadata.labels` field.

    When modifying labels in the running cluster, make sure re-create all the machines to apply the new labels.

    Example:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **nodeGroups.nodeTemplate.taints**
  array of objects

  The same as the `.spec.taints` field of the Node object.

  > **i** Only the `effect`, `key`, and `values` fields are available.

  Example:

  ```
  taints:
  - effect: NoExecute
    key: ship-class
    value: frigate
  ```

  - **nodeGroups.nodeTemplate.taints.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
  - **nodeGroups.nodeTemplate.taints.key**
    string
  - **nodeGroups.nodeTemplate.taints.value**
    string

- **nodeGroups.replicas**
  integer

  Required value

  Number of nodes to create.

- **nodeGroups.zones**
  array of strings

  A set of zones in which nodes can be created.

  To use this setting, the `topology.kubernetes.io/zone` label must be set on DVP nodes. Read more about topological labels.

  > **i** To set the required label for a DVP node, follow the NodeGroup documentation.

- **provider**
  object

  Required value

  Contains settings to connect to the Deckhouse Kubernetes Platform API.

  - **provider.kubeconfigDataBase64**
    string

    Required value

    Kubeconfig to connect to Deckhouse Kubernetes Platform API, Base64-encoded.

    Pattern: `^[A-Za-z0-9+/]+[=]{0,2}$`
  - **provider.namespace**
    string

    Required value

    Namespace in which DKP cluster resources will be created.

    > **i** If not explicitly specified, the default namespace for kubeconfig will be used.

  - **provider.networkPolicy**
    string

    Control rules for network traffic to and from workloads running in the Project resource.

    - `Isolated`: Applies a restrictive NetworkPolicy that allows only network traffic required for

platform system components to function. All other traffic is denied.
  - `None`: The cluster does not request any network policies. Existing project-level restrictions still apply.

Default: `Isolated`

Allowed values: `Isolated`, `None`

- **region**
  string

  Region name.

  To use this setting, the `topology.kubernetes.io/region` label must be set on DVP nodes. Read more about topological labels.

  > **i** To set the required label for a DVP node, follow the NodeGroup documentation.

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **zones**
  array of strings

  A set of zones in which nodes can be created.

  To use this setting, the `topology.kubernetes.io/zone` label must be set on DVP nodes. Read more about topological labels.

  > **i** To set the required label for a DVP node, follow the NodeGroup documentation.

Preparing environment

> **i** The module lifecycle stage: General Availability

Deckhouse Kubernetes Platform components interact with DVP resources through the DVP API. To configure this connection, create a new user (ServiceAccount), assign the necessary permissions, and generate a kubeconfig.

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

> ⚠ If the `update-hostname` module of the `cloud-init` package is not disabled, it is recommended to change its run frequency from `always` to `once-per-instance`.
>
> To do this, modify the `update-hostname` module configuration in the `/etc/cloud/cloud.cfg` file:
>
> ```
> cloud_init_modules:
>   ...
>   - [update-hostname, once-per-instance]
>   ...
> ```
>
> The `update-hostname` module can also be disabled completely by removing it from the `cloud_init_modules` module list in the `/etc/cloud/cloud.cfg` file.

Creating a user

Create a new user in the DVP cluster using the following command:

```
d8 k create -f -<<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-demo
  namespace: default
---
apiVersion: v1
kind: Secret
metadata:
  name: sa-demo-token
  namespace: default
  annotations:
    kubernetes.io/service-account.name: sa-demo
type: kubernetes.io/service-account-token
EOF
```

Adding a role

Add a role to the created user in the DVP cluster using the following command:

```
d8 k create -f -<<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: sa-demo-rb
  namespace: default
subjects:
  - kind: ServiceAccount
    name: sa-demo
    namespace: default
roleRef:
  kind: ClusterRole
  name: d8:use:role:manager
  apiGroup: rbac.authorization.k8s.io
EOF
```

Generating a kubeconfig

Generate a kubeconfig to be used in the cluster initial configuration file:

```
cat <<EOF > kubeconfig
apiVersion: v1
clusters:
- cluster:
    server: https://<KUBE-APISERVER-URL>   # Replace this with the actual API server address
for the cluster.
  name: <CLUSTER-NAME>                     # Replace with the cluster name.
contexts:
- context:
    cluster: <CLUSTER-NAME>                # Replace with the cluster name.
    user: sa-demo
    namespace: default
  name: sa-demo-context
current-context: sa-demo-context
kind: Config
preferences: {}
users:
- name: sa-demo
  user:
    token: $(d8 k get secret sa-demo-token -n default -o json | jq -rc .data.token | base64 -d)
EOF
```

Encode the generated kubeconfig file using Base64 encoding (it appears in the initial configuration file as follows):

```
base64 kubeconfig | tr -d '\n'
```

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

DVPInstanceClass

Scope: Cluster
Version: v1alpha1

- **spec**
  object

  Required value

  - **spec.additionalDisks**
    array of objects

    Parameters for additional virtual machine disks.

    - **spec.additionalDisks.size**
      string

      Required value

      Size of the additional disk.

      Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

      Example:

      ```
      size: 10Gi
      ```

    - **spec.additionalDisks.storageClass**
      string

      Required value

      Name of the existing StorageClass that will be used to create the additional disk.

  - **spec.rootDisk**
    object

    Required value

    Specifies settings for the root disk of the virtual machine.

    - **spec.rootDisk.image**
      object

      Required value

      Image parameters that will be used to create the virtual machine’s root disk.

      - **spec.rootDisk.image.kind**
        string

        Required value

        The kind of the image source.

        Allowed values: `ClusterVirtualImage`, `VirtualImage`, `VirtualDisk`

      - **spec.rootDisk.image.name**
        string

        Required value

        The name of the image that will be used to create the root disk.

        > **i** The installation requires Linux OS images with cloud-init pre-installed.

    - **spec.rootDisk.size**
      string

      Required value

      Root disk size.

      Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

Example:

```
size: 10Gi
```

- **spec.rootDisk.storageClass**
  string

  The name of the existing StorageClass will be used to create the virtual machine's root disk.

  If the value is not specified, the StorageClass will be used according to the global storageClass parameter setting.

- **spec.virtualMachine**
  object

  Required value

  Virtual machine settings for the created node.

  > **i** The `runPolicy: AlwaysOnUnlessStoppedManually` policy is used for virtual machines of nodes. This allows the virtual machine to be stopped manually (for example, for maintenance) without triggering an automatic restart.

  - **spec.virtualMachine.bootloader**
    string

    Defines a bootloader for the virtual machine.

    - `BIOS`: Use BIOS.
    - `EFI`: Use Unified Extensible Firmware (EFI/UEFI).
    - `EFIWithSecureBoot`: Use UEFI/EFI with the Secure Boot support.

    Default: `EFI`

    Allowed values: `BIOS`, `EFI`, `EFIWithSecureBoot`

  - **spec.virtualMachine.cpu**
    object

    Required value

    CPU settings for the virtual machine.

    - **spec.virtualMachine.cpu.coreFraction**
      string

      Guaranteed share of CPU fraction that will be allocated to the virtual machine.

      Default: `100%`

      Pattern: `^100%$|^[1-9][0-9]?%$`

      Example:

      ```
      coreFraction: 100%
      ```

    - **spec.virtualMachine.cpu.cores**
      integer

      Required value

      Number of CPU cores for the virtual machine.

      Allowed values: `1 <= X`

      Example:

      ```
      cores: 4
      ```

  - **spec.virtualMachine.liveMigrationPolicy**
    string

    Live migration policy for the virtual machine.

    - `Manual`: Migration is controlled manually.
    - `Never`: Migration is disabled.
    - `AlwaysSafe`: Always use safe migration (may fail if VM has a high rate of memory changes).
    - `PreferSafe`: Prefer safe migration, fallback to forced if needed.
    - `AlwaysForced`: Always use forced migration with VM slowdown.
    - `PreferForced`: Prefer forced migration (recommended for master nodes due to high memory activity).

    Default: `PreferForced`

    Allowed values: `Manual`, `Never`, `AlwaysSafe`, `PreferSafe`, `AlwaysForced`, `PreferForced`

    Example:

```
liveMigrationPolicy: PreferForced
```

- **spec.virtualMachine.memory**
  object

  Required value

  Specifies the memory settings for the virtual machine.

  - **spec.virtualMachine.memory.size**
    string

    Required value

    Amount of memory resources allowed for the virtual machine.

    Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

    Example:

    ```
    size: 4Gi
    ```

- **spec.virtualMachine.runPolicy**
  string

  Virtual machine run policy.

  - `AlwaysOn`: The virtual machine should always be running.
  - `AlwaysOff`: The virtual machine should always be stopped.
  - `Manual`: The virtual machine state is controlled manually.
  - `AlwaysOnUnlessStoppedManually`: The virtual machine can be stopped manually (for example, for maintenance), but it will automatically start after a host reboot.

  Default: `AlwaysOnUnlessStoppedManually`

  Allowed values: `AlwaysOn`, `AlwaysOff`, `Manual`, `AlwaysOnUnlessStoppedManually`

  Example:

  ```
  runPolicy: AlwaysOnUnlessStoppedManually
  ```

- **spec.virtualMachine.virtualMachineClassName**
  string

  Required value

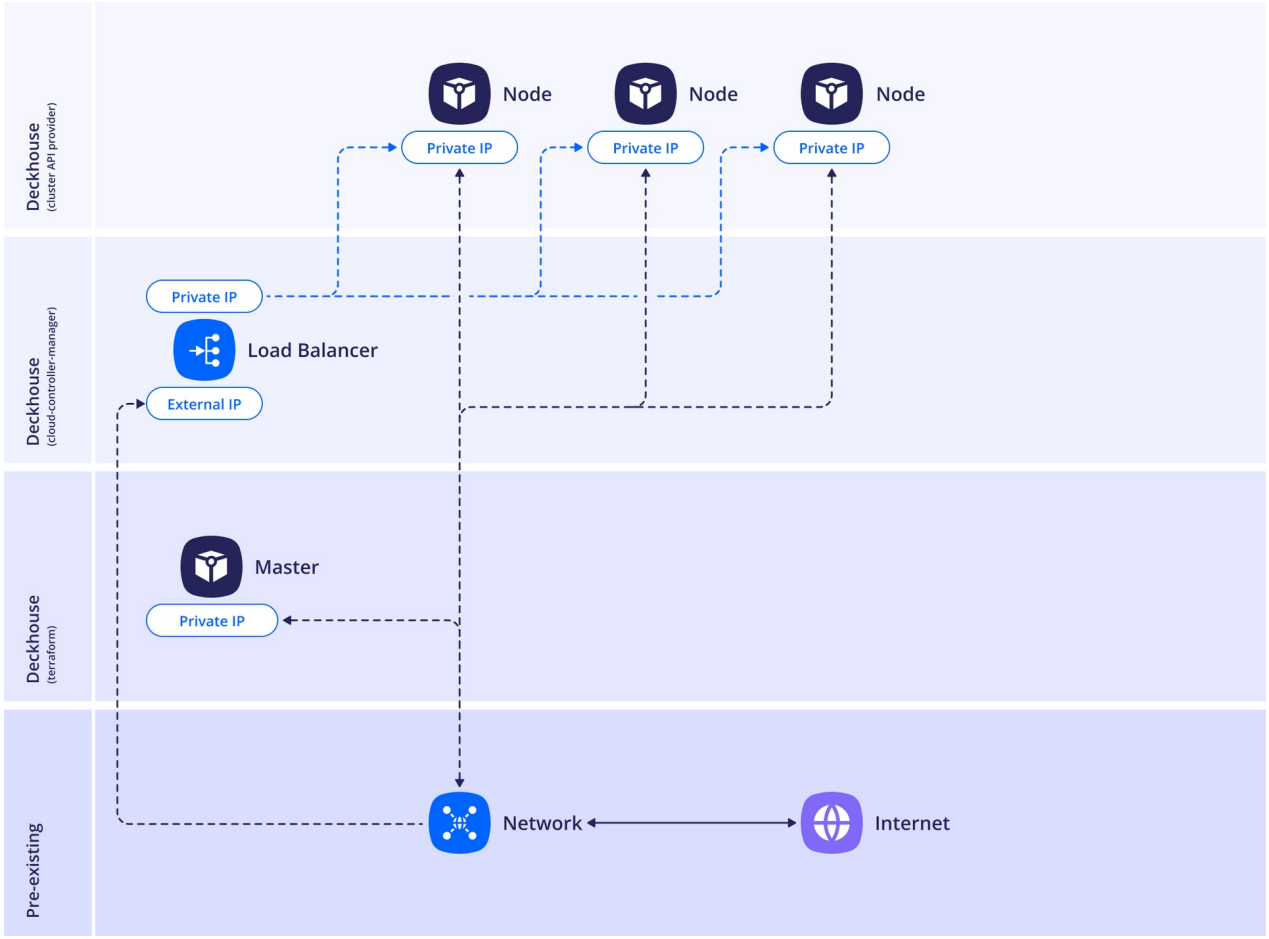
  The name of the VirtualMachineClass.

  Intended for centralized configuration of preferred virtual machine parameters. It allows you to specify CPU instruction sets, resource configuration policies for CPU and memory, and define the ratio between these resources.

Layouts

> **i** The module lifecycle stage: General Availability

Standard



Example of the layout configuration:

```
---
apiVersion: deckhouse.io/v1
kind: DVPClusterConfiguration
layout: Standard
sshPublicKey: <SSH_PUBLIC_KEY>
masterNodeGroup:
  replicas: 1
  instanceClass:
    virtualMachine:
      cpu:
        cores: 4
        coreFraction: 100%
      memory:
        size: 8Gi
      ipAddresses:
        - Auto
      virtualMachineClassName: generic
    rootDisk:
      size: 50Gi
      storageClass: ceph-pool-r2-csi-rbd-immediate
      image:
        kind: ClusterVirtualImage
        name: ubuntu-2204
    etcdDisk:
      size: 15Gi
      storageClass: ceph-pool-r2-csi-rbd-immediate
provider:
  kubeconfigDataBase64: <KUBE_CONFIG>
  namespace: demo
```

Cloud provider — Basis Dynamix

⚠ The module lifecycle stage: Preview

The `cloud-provider-dynamix` module integrates Deckhouse Kubernetes Platform with the Basis Dynamix platform. It allows the `node-manager` module to use Dynamix resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-dynamix` module:

- Managing Dynamix resources via `cloud-controller-manager`:
  - updates virtual machine and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in Dynamix;
  - creates load balancers for Services of the LoadBalancer type. The Service must include annotations with the names of the internal and external networks.
- Provisioning disks via the Dynamix CSI driver (`dynamix.deckhouse.io`) so that PersistentVolumes can be requested from the cluster.
- Provisioning CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-decort`.
- Provisioning CloudEphemeral nodes via Cluster API (CAPI). Virtual machine parameters are set in the DynamixInstanceClass resource.
- Registering with `node-manager` so that DynamixInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

> **i** The module is in the Preview stage.

Configuration

> **⚠** The module lifecycle stage: Preview

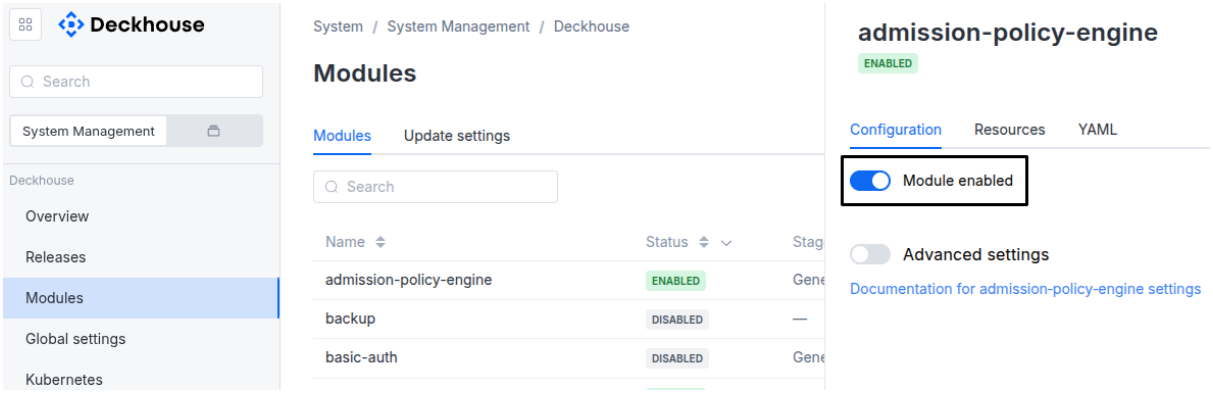
The module is automatically enabled for all cloud clusters deployed in Basis Dynamix.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-dynamix` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cloud-provider-dynamix
```

- **Using ModuleConfig `cloud-provider-dynamix`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-dynamix` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-dynamix`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-dynamix
spec:
  enabled: true
```

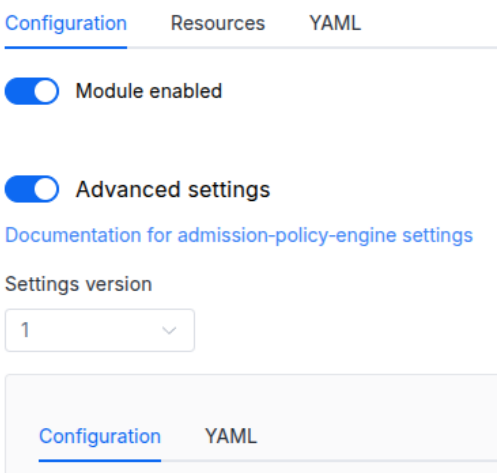
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-dynamix` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

Example:



You can also edit the ModuleConfig object `cloud-provider-dynamix` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-dynamix`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-dynamix` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-dynamix
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cloud-provider-dynamix` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cloud-provider-dynamix`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-dynamix
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

The module does not have any settings.

Provider configuration

⚠ The module lifecycle stage: Preview

i If the cluster control plane is hosted on a virtual machines or bare-metal servers, the cloud provider uses the settings from the `cloud-provider-dynamix` module in the Deckhouse configuration. If the cluster control plane is hosted in a cloud, the cloud provider uses the DynamixClusterConfiguration structure for configuration.

DynamixClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in Dynamix.

Used by the cloud provider if a cluster’s control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

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

> **i** After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: StandardWithInternalNetwork
sshPublicKey: ssh-rsa AAAA
location: dynamix
account: acc_user
nodeNetworkCIDR: 10.241.32.0/24
nameservers:
- 10.0.0.10
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    storageEndpoint: "<storage endpoint>"
    pool: "<pool>"
    externalNetwork: "<external network>"
nodeGroups:
- name: worker
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    externalNetwork: "<external network>"
```

- **account**
  string

  Required value

  Account name.

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`

- **kind**
  string

  Required value

  Allowed values: `DynamixClusterConfiguration`

- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

- **location**
  string

  Required value

  The name of the location to install the cluster.

- **masterNodeGroup**
  object

Required value

The definition of the master’s NodeGroup.

> **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

- **masterNodeGroup.instanceClass**
  object

  Required value

  Partial contents of the fields of the DynamixInstanceClass.

  - **masterNodeGroup.instanceClass.etcdDiskSizeGb**
    integer

    Etcd disk size in GiB.

    Default: `15`

    Example:

    ```
    etcdDiskSizeGb: 15
    ```

  - **masterNodeGroup.instanceClass.externalNetwork**
    string

    Required value

    External network name.

    Example:

    ```
    externalNetwork: extnet_vlan_1700
    ```

  - **masterNodeGroup.instanceClass.imageName**
    string

    Required value

    OS image name to use for VM deployment.

    Example:

    ```
    imageName: '1'
    ```

  - **masterNodeGroup.instanceClass.memory**
    integer

    Required value

    Memory in MiB to allocate to the VM.

    Allowed values: `1 <= X`

    Example:

    ```
    memory: 8192
    ```

  - **masterNodeGroup.instanceClass.numCPUs**
    integer

    Required value

    Number of vCPUs to allocate to the VM.

    Allowed values: `1 <= X`

    Example:

    ```
    numCPUs: 2
    ```

  - **masterNodeGroup.instanceClass.pool**
    string

    Required value

    Storage pool name.

    Example:

    ```
    pool: pool_a
    ```

  - **masterNodeGroup.instanceClass.rootDiskSizeGb**

integer

Root disk size in GiB to use in Dynamix VirtualMachines.

Default: `50`

Example:

```
rootDiskSizeGb: 50
```

- **masterNodeGroup.instanceClass.storageEndpoint**
  string

  Required value

  Storage name.

  Example:

  ```
  storageEndpoint: SharedTatlin_G1_SEP
  ```

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **nameservers**
  array of strings

  A list of IP addresses of the DNS servers.

  Default: `[]`

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the.

    - **nodeGroups.instanceClass.etcdDiskSizeGb**
      integer

      Etcd disk size in GiB.

      Default: `15`

      Example:

      ```
      etcdDiskSizeGb: 15
      ```

    - **nodeGroups.instanceClass.externalNetwork**
      string

      Required value

      External network name.

      Example:

      ```
      externalNetwork: extnet_vlan_1700
      ```

    - **nodeGroups.instanceClass.imageName**
      string

      Required value

      OS image name to use for VM deployment.

      Example:

      ```
      imageName: '1'
      ```

    - **nodeGroups.instanceClass.memory**
      integer

      Required value

      Memory in MiB to allocate to the VM.

Allowed values: `1 <= X`

Example:

```
memory: 8192
```

- **nodeGroups.instanceClass.numCPUs**
integer

  Required value

  Number of vCPUs to allocate to the VM.

  Allowed values: `1 <= X`

  Example:

```
numCPUs: 2
```

- **nodeGroups.instanceClass.pool**
string

  Storage pool name.

  Example:

```
pool: pool_a
```

- **nodeGroups.instanceClass.rootDiskSizeGb**
integer

  Root disk size in GiB to use in Dynamix VirtualMachines.

  Default: `50`

  Example:

```
rootDiskSizeGb: 50
```

- **nodeGroups.instanceClass.storageEndpoint**
string

  Storage name.

  Example:

```
storageEndpoint: SharedTatlin_G1_SEP
```

- **nodeGroups.name**
string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
object

    The same as the `metadata.annotations` standard field.

    Example:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

  - **nodeGroups.nodeTemplate.labels**
object

    A list of labels to attach to cluster resources.

    The same as the `metadata.labels` standard field.

    Example:

```
labels:
  environment: production
  app: warp-drive-ai
```

  - **nodeGroups.nodeTemplate.taints**
array of objects

The same as the `.spec.taints` field of the Node object.

> **i** Available fields: `effect`, `key`, and `values`.

Example:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

- **nodeGroups.nodeTemplate.taints.effect**
  string

  Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
- **nodeGroups.nodeTemplate.taints.key**
  string
- **nodeGroups.nodeTemplate.taints.value**
  string

- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes to create.

- **nodeNetworkCIDR**
  string

  IP CIDR to be assigned to the internal segment of this ViNS.

- **provider**
  object

  Required value

  Parameters for connecting to the Dynamix.

  - **provider.appId**
    string

    Required value

    App ID.

  - **provider.appSecret**
    string

    Required value

    App Secret.

  - **provider.controllerUrl**
    string

    Required value

    The URL to the Dynamix API endpoint.

  - **provider.insecure**
    boolean

    Set to `true` if Dynamix has a self-signed certificate.

    Default: `false`

  - **provider.oAuth2Url**
    string

    Required value

    The URL to the SSO Dynamix API endpoint.

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

Preparing environment

> **⚠** The module lifecycle stage: Preview

Prepare an operating system image

Operating system vendors typically provide special cloud builds of their operating systems for use in virtualization environments. These builds typically contain virtual hardware drivers, cloud-init, virtualization guest agents, and are distributed as IMG or QCOW2 disk images. We recommend that you use these cloud images as the OS on the nodes in your clusters.

The cloud image of the operating system must be placed in the “Images” → “Template Images” section of the Basis Dynamix portal. Follow these steps to upload the OS image to the storage:

If the infrastructure does not have a DNS server, access to the Basis Dynamix portal from the cluster. Basis Dynamix portal from the cluster can be organized by adding the IP address and domains associated with the Basis Dynamix portal to the `cloud-init` template for generating the hosts file. This template is located in the `/etc/cloud/templates/` folder. The name of the template depends on the OS.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

⚠ After adding data to hosts and before creating the template from the virtual machine, you must run the `cloud-init clean` command.

Custom Resources

ℹ The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

⚠ The module lifecycle stage: Preview

DynamixInstanceClass

Scope: Cluster
Version: v1

Parameters of a group of Dynamix VirtualMachines used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.externalNetwork**
    string

    External network name.

    Example:

    ```
    externalNetwork: extnet_vlan_1700
    ```

  - **spec.imageName**
    string

    Required value

    Name of the OS image to base this VM on.

    Example:

    ```
    imageName: alt-p10-cloud-x86_64.img
    ```

  - **spec.memory**
    integer

    Required value

    Memory in MiB to allocate to Dynamix VirtualMachines.

    Example:

    ```
    memory: 8192
    ```

  - **spec.numCPUs**
    integer

    Required value

Count of vCPUs to allocate to Dynamix VirtualMachines.

Example:

```
numCPUs: 2
```

- **spec.rootDiskSizeGb**
  integer

  Root disk size in GiB to use in Dynamix VirtualMachines.

  The disk will be automatically resized if its size in the template differs from specified.

  Default: `30`

  Example:

  ```
  rootDiskSizeGb: 30
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

FAQ

⚠ The module lifecycle stage: Preview

How to configure a LoadBalancer?

o configure a Service of the LoadBalancer type, add the following annotations to the Service manifest:

```
metadata:
  annotations:
    dynamix.cpi.flant.com/internal-network-name: <internal_name>
    dynamix.cpi.flant.com/external-network-name: <external_name>
```

Both annotations are required:

- `dynamix.cpi.flant.com/internal-network-name` — the name of the internal network in Basis Dynamix
- `dynamix.cpi.flant.com/external-network-name` — the name of the external network in Basis Dynamix

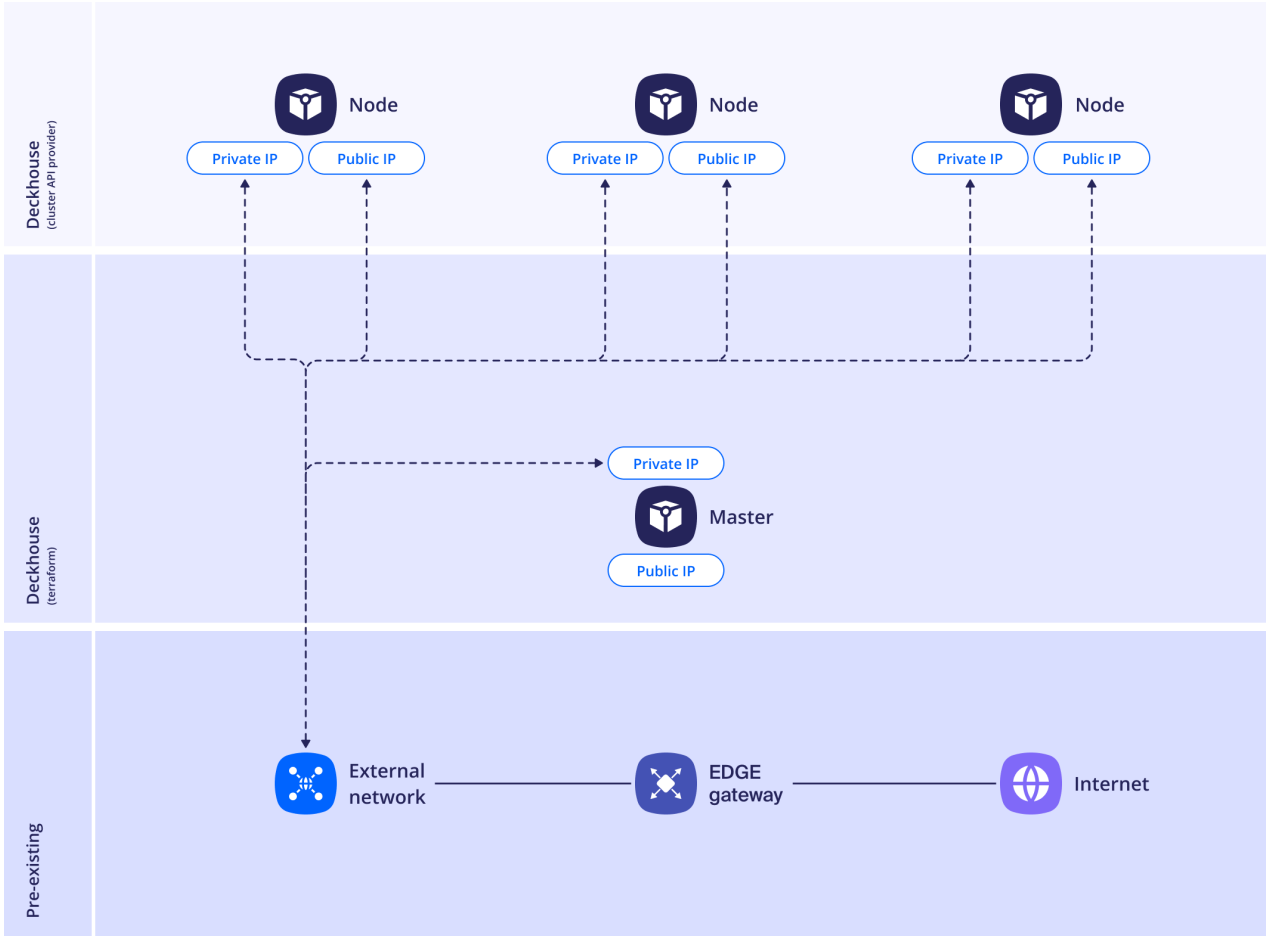
The terms “internal network” and “external network” are used in the context of Basis Dynamix. The external network does not have to be public and may use private IP addresses.

If one of the annotations is not specified, cloud-controller-manager will fail to process the Service.

Cloud provider - Basis Dynamix: Layouts

⚠ The module lifecycle stage: Preview

Standard



Example of the layout configuration:

```
---
apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
location: dynamix
account: acc_user
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    storageEndpoint: "<storage endpoint>"
    pool: "<pool>"
    externalNetwork: "<external network>"
```

StandardWithInternalNetwork

Example of the layout configuration:

```
---
apiVersion: deckhouse.io/v1
kind: DynamixClusterConfiguration
layout: StandardWithInternalNetwork
sshPublicKey: "<SSH_PUBLIC_KEY>"
location: dynamix
account: acc_user
nodeNetworkCIDR: "10.241.32.0/24"
nameservers: ["10.0.0.10"]
provider:
  controllerUrl: "<controller url>"
  oAuth2Url: "<oAuth2 url>"
  appId: "<app id>"
  appSecret: "<app secret>"
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 6
    memory: 16384
    rootDiskSizeGb: 50
    imageName: "<image name>"
    storageEndpoint: "<storage endpoint>"
    pool: "<pool>"
    externalNetwork: "<external network>"
```

Cloud provider — GCP

> **i** The module lifecycle stage: General Availability

The `cloud-provider-gcp` module integrates Deckhouse Kubernetes Platform with Google Cloud Platform (GCP). It allows the `node-manager` module to use GCP resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-gcp` module:

- Managing GCP resources via `cloud-controller-manager`:
  - creates network routes for the `PodNetwork` network on the GCP side;
  - creates load balancers for Services of the LoadBalancer type;
  - updates cluster node metadata and removes from Kubernetes nodes that no longer exist in GCP.
- Provisioning disks via the Persistent Disk CSI driver (`pd.csi.storage.gke.io`) and creating StorageClasses for GCP disk types so that PersistentVolumes can be requested from the cluster.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-google`.
- Provisioning CloudEphemeral nodes via Machine Controller Manager (MCM). Virtual machine parameters are set in the GCPInstanceClass resource.
- Registering with `node-manager` so that GCPInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

Configuration

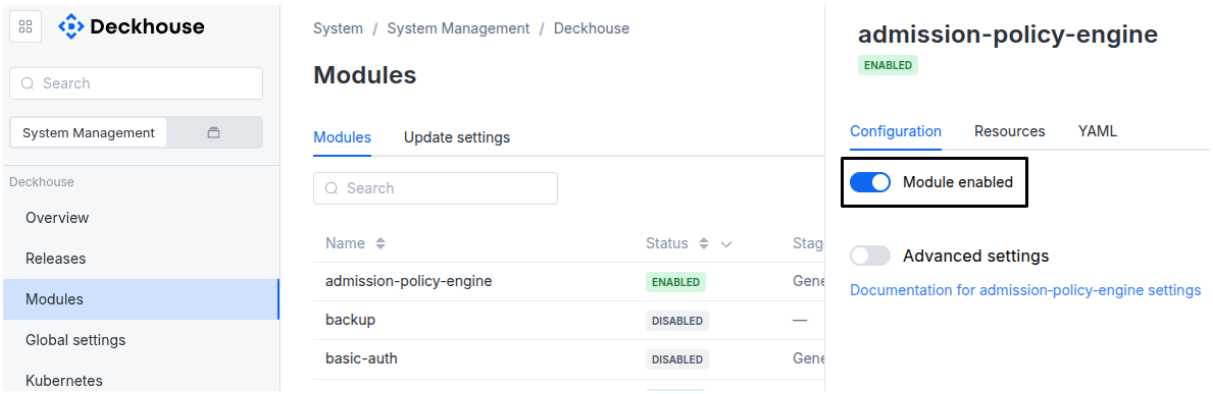
> **i** The module lifecycle stage: General Availability

The module is configured automatically based on the chosen placement strategy defined in the GCPClusterConfiguration struct. In most cases, you do not have to configure the module manually.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-gcp` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable cloud-provider-gcp
  ```

- **Using ModuleConfig `cloud-provider-gcp`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-gcp` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-gcp`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-gcp
  spec:
    enabled: true
  ```

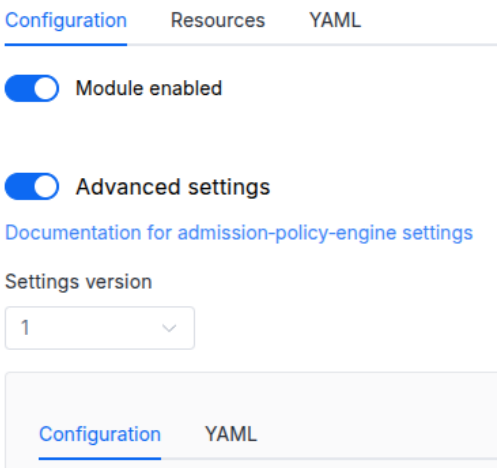
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-gcp` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `cloud-provider-gcp` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-gcp`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-gcp` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-gcp
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the

`spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `cloud-provider-gcp` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `cloud-provider-gcp`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-gcp
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

You can configure the number and parameters of provisioning machines in the cloud via the `NodeGroup` custom resource of the node-manager module. Also, in this custom resource, you can specify the instance class's name for the above group of nodes (the `cloudInstances.ClassReference` parameter of NodeGroup). In the case of the GCP cloud provider, the instance class is the `GCPInstanceClass` custom resource that stores specific parameters of the machines.

The module automatically creates StorageClasses that cover all the available disk types in GCP:

Type	Replication	StorageClass Name
standard	none	pd-standard-not-replicated
standard	regional	pd-standard-replicated
balanced	none	pd-balanced-not-replicated
balanced	regional	pd-balanced-replicated
ssd	none	pd-ssd-not-replicated
ssd	regional	pd-ssd-replicated

Also, it can filter out the unnecessary StorageClasses (you can do this via the `exclude` parameter).

Parameters

Schema version: 1

- **settings**
  object
  - **settings.storageClass**
    object
    - **settings.storageClass.default**
      Deprecated
      string

      The name of StorageClass that will be used in the cluster by default.

      If the parameter is omitted, the default StorageClass is either:

      - existing StorageClass present in the cluster that has the default annotation;
      - the first (in lexicographic order) StorageClass created by the module (in accordance with the order listed in the table above).

      > **i** **Parameter is deprecated.** Instead, use the global parameter global.defaultClusterStorageClass.

      Example:

      ```
      default: pd-ssd-not-replicated
      ```

    - **settings.storageClass.exclude**
      array of strings

      A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster;

      Example:

      ```
      exclude:
      - pd-standard.*
      - pd-ssd-replicated
      ```

Provider configuration

> **i** The module lifecycle stage: General Availability

GCPClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in GCP.

Used by the cloud provider if a cluster's control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

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

> **i** After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
sshKey: "<SSH_PUBLIC_KEY>"
subnetworkCIDR: 10.36.0.0/24
masterNodeGroup:
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
    diskSizeGb: 50
nodeGroups:
- name: static
  replicas: 1
  zones:
  - europe-west3-b
  instanceClass:
    machineType: n1-standard-4
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
    diskSizeGb: 50
    additionalNetworkTags:
    - tag1
    additionalLabels:
      kube-node: static
provider:
  region: europe-west3
  serviceAccountJSON: "<SERVICE_ACCOUNT_JSON>"
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`
- **kind**
  string

  Required value

  Allowed values: `GCPClusterConfiguration`
- **labels**
  object

  A list of labels to attach to cluster resources.

  Note that you have to re-create all the machines to add new tags if tags were modified in the running cluster.

  You can learn more about the labels in the official documentation.

  Format — `key: value`.
- **layout**

string

Required value

The way resources are located in the cloud.

`Standard` - set Cloud NAT mode. More info...

`WithoutNAT` - a dedicated VPC is created for the cluster. All cluster nodes have public IP addresses. More info...

Allowed values: `Standard`, `WithoutNAT`

- **masterNodeGroup**
  object

  Required value

  Parameters of the master's NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the GCPInstanceClass fields.

    - **masterNodeGroup.instanceClass.additionalLabels**
      object

      Additional labels.

      For details on labels, refer to the Google Cloud documentation.

      Example:

      ```
      project: cms-production
      severity: critical
      ```

    - **masterNodeGroup.instanceClass.additionalNetworkTags**
      array of strings

      The list of additional network tags.

      You can use network tags to apply firewall rules to instances. The detailed description of network tags is available in the Google Cloud documentation.

    - **masterNodeGroup.instanceClass.disableExternalIP**
      boolean

      Defines whether to disable external IP for an instance or not.

      This parameter is only available for the `Standard` layout.

      True means that nodes do not have public addresses and connect to the Internet over `CloudNAT`;

      False means that static public addresses are created for nodes, they are also used for One-to-one NAT.;

      Default: `true`

      Allowed values: `true`, `false`

    - **masterNodeGroup.instanceClass.diskSizeGb**
      integer

      Instance disk size in gibibytes.

      Example:

      ```
      diskSizeGb: 50
      ```

    - **masterNodeGroup.instanceClass.etcdDiskSizeGb**
      integer

      Etcd disk size in gibibytes.

      Default: `20`

      Example:

      ```
      etcdDiskSizeGb: 20
      ```

    - **masterNodeGroup.instanceClass.image**
      string

      Required value

Image to use while provisioning GCP servers.

You can find a list of available images in the provider documentation.

The list of OS and their versions supported by Deckhouse can be found in the Deckhouse documentation (take into account the Deckhouse version used).

Example:

```
image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
```

- **masterNodeGroup.instanceClass.machineType**
  string

  Required value

  Machine type of GCP instance.

  > **Caution!** Make sure that this type is present in all zones specified in the `zones` parameter.

  GCP lets you specify a custom amount of resources (CPU and RAM), e.g., `custom-8-40960` or `n2-custom-8-40960`.

  Example:

  ```
  machineType: n1-standard-4
  ```

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create.

  It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.zones**
  array of strings

  A limited set of zones in which master nodes can be created.

  The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the GCPInstanceClass fields.

    - **nodeGroups.instanceClass.additionalLabels**
      object

      Additional labels.

      For details on labels, refer to the Google Cloud documentation.

      Example:

      ```
      project: cms-production
      severity: critical
      ```

    - **nodeGroups.instanceClass.additionalNetworkTags**
      array of strings

      The list of additional network tags.

      You can use network tags to apply firewall rules to instances. The detailed description of network tags is available in the Google Cloud documentation.

    - **nodeGroups.instanceClass.disableExternalIP**
      boolean

      Defines whether to disable external IP for an instance or not.

      This parameter is only available for the `Standard` layout.

True means that nodes do not have public addresses and connect to the Internet over `CloudNAT`;

False means that static public addresses are created for nodes, they are also used for One-to-one NAT.;

Default: `true`

Allowed values: `true`, `false`

- **nodeGroups.instanceClass.diskSizeGb**
  integer

  Instance disk size in gibibytes.

  Example:

  ```
  diskSizeGb: 50
  ```

- **nodeGroups.instanceClass.image**
  string

  Required value

  Image to use while provisioning GCP servers.

  You can find a list of available images in the provider documentation.

  The list of OS and their versions supported by Deckhouse can be found in the Deckhouse documentation (take into account the Deckhouse version used).

  Example:

  ```
  image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
  ```

- **nodeGroups.instanceClass.machineType**
  string

  Required value

  Machine type of GCP instance.

  > **Caution!** Make sure that this type is present in all zones specified in the `zones` parameter.

  GCP lets you specify a custom amount of resources (CPU and RAM), e.g., `custom-8-40960` or `n2-custom-8-40960`.

  Example:

  ```
  machineType: n1-standard-4
  ```

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**
  object

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta).

  - **nodeGroups.nodeTemplate.labels**
    object

    The same as the `metadata.labels` standard (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta).

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object(https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#taint-v1-core).

    > Available fields: `effect`, `key`, and `values`.

    - **nodeGroups.nodeTemplate.taints.effect**
      string

      Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

        - **nodeGroups.nodeTemplate.taints.key**
          string
        - **nodeGroups.nodeTemplate.taints.value**
          string
    - **nodeGroups.replicas**
      integer

      Required value

      The number of nodes to create.

    - **nodeGroups.zones**
      array of strings

      A limited set of zones in which master nodes can be created.

      The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

- **peeredVPCs**
  array of strings

  A list of GCP VPC networks to peer with the cluster network.

  The service account must have access to all the VPCs listed. You have to configure the peering connection manually if no access is available.

- **provider**
  object

  Required value

  Parameters for connecting to the GCP API.

    - **provider.region**
      string

      Required value

      The name of the region where instances will be provisioned.

    - **provider.serviceAccountJSON**
      string

      Required value

      A key to the Service Account with Project Admin privileges (`service account key`) in the JSON format.

      How to create it a `service account key`.

- **sshAllowList**
  array of strings

  A list of CIDR’s allowed to connect to nodes via SSH.

  By default, from any.

- **sshKey**
  string

  Required value

  A public key to access nodes as `user`.

- **standard**
  object

  Settings for the `Standard` layout.

    - **standard.cloudNATAddresses**
      array of strings

      A list of public static IP addresses for `Cloud NAT`.

- **subnetworkCIDR**
  string

  A subnet to use for cluster nodes.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **withoutNAT**
  object

  Settings for the `WithoutNAT` layout.

- **zones**
  array of strings

  A limited set of zones in which master nodes can be created.

The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

Preparing environment

ℹ The module lifecycle stage: General Availability

⚠ To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:

- Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- `CgroupsV2` support.
- Systemd version `244` or newer.
- `erofs` kernel module support.

For more information, see the documentation.

⚠ The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

You need to create a service account so that Deckhouse Kubernetes Platform can manage resources in the Google Cloud. Below is a brief sequence of steps to create a service account. If you need detailed instructions, you can find them in the provider's documentation.

⚠ **Note!** The created `service account key` cannot be restored, you can only delete and create a new one.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Setup using Google Cloud Console

Follow this link, select your project and create a new service account or select an existing one.

The account must be assigned several necessary roles:

```
Compute Admin
Service Account User
Network Management Admin
```

You can add roles when creating a service account or edit them here.

To create a `service account key` in JSON format, click on three vertical dots in the Actions column and select `Manage keys`. Next, click on `Add key` -> `Create new key` -> `Key type` -> `JSON`.

Setup using gcloud CLI

To configure via the command line interface, follow these steps:

1. Export environment variables:

   ```
   export PROJECT_ID=sandbox
   export SERVICE_ACCOUNT_NAME=deckhouse
   ```

2. Select a project:

   ```
   gcloud config set project $PROJECT_ID
   ```

3. Create a service account:

   ```
   gcloud iam service-accounts create $SERVICE_ACCOUNT_NAME
   ```

4. Connect roles to the service account:

```
for role in roles/compute.admin roles/iam.serviceAccountUser roles/networkmanagement.admin;
do gcloud projects add-iam-policy-binding ${PROJECT_ID} --
member=serviceAccount:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
   --role=${role}; done
```

List of roles required:

```
roles/compute.admin
roles/iam.serviceAccountUser
roles/networkmanagement.admin
```

5. Verify service account roles:

```
gcloud projects get-iam-policy ${PROJECT_ID} --flatten="bindings[].members" --
format='table(bindings.role)' \
     --filter="bindings.members:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com"
```

6. Create a `service account key`:

```
gcloud iam service-accounts keys create --iam-account
${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
     ~/service-account-key-${PROJECT_ID}-${SERVICE_ACCOUNT_NAME}.json
```

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

GCPInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of a group of GCP Instances used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalDisks**
    array of objects

    A list of additional disks to attach to the instance.

    Useful for storage workloads such as LINSTOR, Ceph, NFS, and similar.

    Example:

    ```
    additionalDisks:
    - size: 100
      type: pd-ssd
    ```

    - **spec.additionalDisks.autoDelete**
      boolean

      Automatically delete the disk when the instance is deleted.

      Default: `true`

    - **spec.additionalDisks.provisionedIops**
      integer

      Provisioned IOPS for the disk.

      Required for `pd-extreme` and `hyperdisk-extreme` disk types.

      Allowed values: `1 <= X`

    - **spec.additionalDisks.provisionedThroughput**
      integer

Provisioned throughput in MiB/s for the disk.

Applicable for `hyperdisk-balanced` and `hyperdisk-throughput` disk types.

Allowed values: `1 <= X`

- **spec.additionalDisks.size**
  integer

  Required value

  Disk size in gigabytes (GB).

  Allowed values: `1 <= X`

  Example:

  ```
  size: 100
  ```

- **spec.additionalDisks.type**
  string

  Required value

  Disk type.

  - `pd-standard` — standard Persistent Disk (HDD).
  - `pd-balanced` — balanced Persistent Disk (SSD).
  - `pd-ssd` — SSD Persistent Disk.
  - `pd-extreme` — extreme Persistent Disk; requires `provisionedIops`.
  - `hyperdisk-balanced` — Hyperdisk with balanced performance.
  - `hyperdisk-throughput` — Hyperdisk optimized for throughput.
  - `hyperdisk-extreme` — Hyperdisk with maximum performance; requires `provisionedIops`.

  Allowed values: `pd-standard`, `pd-balanced`, `pd-ssd`, `pd-extreme`, `hyperdisk-balanced`, `hyperdisk-throughput`, `hyperdisk-extreme`

  Example:

  ```
  type: pd-ssd
  ```

- **spec.additionalLabels**
  object

  Additional labels.

  More info...

  Example:

  ```
  additionalLabels:
    project: cms-production
    severity: critical
  ```

- **spec.additionalNetworkTags**
  array of strings

  The list of additional tags.

  For example, you can use tags to apply firewall rules to instances. The detailed description of network tags is available in the official documentation.

- **spec.capacity**
  Deprecated
  object

  Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

  Instance capacity for *non-standard* instance types (the `machineType` parameter).

  Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

  The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`a2-highgpu-8g`, `c2-standard-8`, etc.) is embedded in Deckhouse.

  - **spec.capacity.cpu**
    integer or string

    Required value

    Node vCPU resources.

    Pattern: `^[0-9]+m?$`

    Example:

```
cpu: 1000m
```

- **spec.capacity.memory**
  integer or string

  Required value

  Node memory resources.

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  Example:

  ```
  memory: 1500Mi
  ```

- **spec.disableExternalIP**
  boolean

  Defines whether to disable external ip for an instance or not. True means that an instance is created without external ip

- **spec.diskSizeGb**
  integer

  Instance’s root disk size in gibibytes.

  Default: `50`

  Example:

  ```
  diskSizeGb: 20
  ```

- **spec.diskType**
  string

  Instance’s disk type.

  Default: `pd-standard`

  Allowed values: `pd-standard`, `pd-ssd`

  Example:

  ```
  diskType: pd-ssd
  ```

- **spec.enableNestedVirtualization**
  boolean

  Enables nested virtualization on the instance.

  Required for running virtual machine workloads (e.g., KVM-based VMs) inside GCP instances.

  Warning. Only supported on specific machine types. See the GCP documentation for the list of compatible types.

  Default: `false`

- **spec.image**
  string

  Not required value.

  Image to use while provisioning GCP servers.

  You can find a list of available images in the provider documentation.

  The list of OS and their versions supported by Deckhouse can be found in the Deckhouse documentation (take into account the Deckhouse version used).

  Example:

  ```
  image: https://www.googleapis.com/compute/v1/projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
  ```

- **spec.machineType**
  string

  Required value

  Machine type of GCP instance.

  Caution! Make sure that this type is present in all zones specified in the `zones` parameter.

  GCP lets you specify a custom amount of resources (CPU and RAM), e.g., `custom-8-40960` or `n2-custom-8-40960`.

  Example:

```
machineType: n1-standard-4
```

  - **spec.preemptible**
    boolean

    Should instances be preemptible.

    Default: `false`

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Parameters of a group of GCP Instances used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalDisks**
    array of objects

    A list of additional disks to attach to the instance.

    Useful for storage workloads such as LINSTOR, Ceph, NFS, and similar.

    Example:

    ```
    additionalDisks:
    - size: 100
      type: pd-ssd
    ```

    - **spec.additionalDisks.autoDelete**
      boolean

      Automatically delete the disk when the instance is deleted.

      Default: `true`

    - **spec.additionalDisks.provisionedIops**
      integer

      Provisioned IOPS for the disk.

      Required for `pd-extreme` and `hyperdisk-extreme` disk types.

      Allowed values: `1 <= X`

    - **spec.additionalDisks.provisionedThroughput**
      integer

      Provisioned throughput in MiB/s for the disk.

      Applicable for `hyperdisk-balanced` and `hyperdisk-throughput` disk types.

      Allowed values: `1 <= X`

    - **spec.additionalDisks.size**
      integer

      Required value

      Disk size in gigabytes (GB).

      Allowed values: `1 <= X`

      Example:

      ```
      size: 100
      ```

    - **spec.additionalDisks.type**
      string

      Required value

      Disk type.

      - `pd-standard` — standard Persistent Disk (HDD).
      - `pd-balanced` — balanced Persistent Disk (SSD).
      - `pd-ssd` — SSD Persistent Disk.
      - `pd-extreme` — extreme Persistent Disk; requires `provisionedIops`.
      - `hyperdisk-balanced` — Hyperdisk with balanced performance.
      - `hyperdisk-throughput` — Hyperdisk optimized for throughput.
      - `hyperdisk-extreme` — Hyperdisk with maximum performance; requires `provisionedIops`.

Allowed values: `pd-standard`, `pd-balanced`, `pd-ssd`, `pd-extreme`, `hyperdisk-balanced`, `hyperdisk-throughput`, `hyperdisk-extreme`

Example:

```
type: pd-ssd
```

- **spec.additionalLabels**
  object

  Additional labels.

  More info...

  Example:

  ```
  additionalLabels:
    project: cms-production
    severity: critical
  ```

- **spec.additionalNetworkTags**
  array of strings

  The list of additional tags.

  For example, you can use tags to apply firewall rules to instances. The detailed description of network tags is available in the official documentation.

- **spec.capacity**
  Deprecated
  object

  Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

  Instance capacity for *non-standard* instance types (the `machineType` parameter).

  Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

  The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`a2-highgpu-8g`, `c2-standard-8`, etc.) is embedded in Deckhouse.

  - **spec.capacity.cpu**
    integer or string

    Required value

    Node vCPU resources.

    Pattern: `^[0-9]+m?$`

    Example:

    ```
    cpu: 1000m
    ```

  - **spec.capacity.memory**
    integer or string

    Required value

    Node memory resources.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    Example:

    ```
    memory: 1500Mi
    ```

- **spec.disableExternalIP**
  boolean

  Defines whether to disable external ip for an instance or not. True means that an instance is created without external ip

- **spec.diskSizeGb**
  integer

  Instance’s root disk size in gibibytes.

  Default: `50`

  Example:

  ```
  diskSizeGb: 20
  ```

- **spec.diskType**
  string

  Instance’s disk type.

  Default: `pd-standard`

  Allowed values: `pd-standard`, `pd-ssd`

  Example:

  ```
  diskType: pd-ssd
  ```

- **spec.enableNestedVirtualization**
  boolean

  Enables nested virtualization on the instance.

  Required for running virtual machine workloads (e.g., KVM-based VMs) inside GCP instances.

  Warning. Only supported on specific machine types. See the GCP documentation for the list of compatible types.

  Default: `false`

- **spec.image**
  string

  Not required value.

  Image to use while provisioning GCP servers.

  You can find a list of available images in the provider documentation.

  The list of OS and their versions supported by Deckhouse can be found in the Deckhouse documentation (take into account the Deckhouse version used).

  Example:

  ```
  image: https://www.googleapis.com/compute/v1/projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128
  ```

- **spec.machineType**
  string

  Required value

  Machine type of GCP instance.

  Caution! Make sure that this type is present in all zones specified in the `zones` parameter.

  GCP lets you specify a custom amount of resources (CPU and RAM), e.g., `custom-8-40960` or `n2-custom-8-40960`.

  Example:

  ```
  machineType: n1-standard-4
  ```

- **spec.preemptible**
  boolean

  Should instances be preemptible.

  Default: `false`

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

FAQ

> **i** The module lifecycle stage: General Availability

How do I create a cluster?

1. Set up a cloud environment.
2. Enable the module or pass the `--extra-config-map-data base64_encoding_of_custom_config` flag with the module parameters to the `install.sh` script.
3. Create one or more GCPInstanceClass custom resources.
4. Create one or more NodeGroup custom resources for managing the number and the process of provisioning machines in the cloud.

Adding CloudStatic nodes to a cluster

For the VMs, you want to add to a cluster as nodes, add a `Network Tag` similar to the cluster prefix.

You can find out `prefix` using the command:

```
d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r '.data."cluster-configuration.yaml"' \
  | base64 -d | grep prefix
```

Examples

> **i** The module lifecycle stage: General Availability

An example of the `GCPInstanceClass` custom resource

Below is a simple example of custom resource `GCPInstanceClass` configuration:

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: test
spec:
  machineType: n1-standard-1
```

Enabling nested virtualization

To run virtual machine workloads (e.g., KVM-based VMs) inside GCP instances, enable nested virtualization.

> **i** Only specific machine types support nested virtualization. See the GCP documentation for the list of compatible types.

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: vm-nodes
spec:
  machineType: n2-standard-8
  enableNestedVirtualization: true
```

Adding additional disks

To attach additional disks to instances (for example, for LINSTOR, Ceph, NFS storage nodes, and similar solutions), specify them in the `additionalDisks` parameter:

```
apiVersion: deckhouse.io/v1
kind: GCPInstanceClass
metadata:
  name: storage-nodes
spec:
  machineType: n1-standard-8
  additionalDisks:
  - size: 200
    type: pd-ssd
  - size: 500
    type: pd-standard
    autoDelete: true
```

Configuring security policies on nodes

For cluster virtual machines in GCP, you may need to limit or expand incoming and outgoing traffic for various reasons. Some of these may include:

- Allowing connections to cluster nodes from virtual machines from a different subnet.
- Allow connections to static node ports for application operation.
- Restricting access to external resources or other virtual machines in the cloud at the request of the security service.

For all this, additional network tags must be used.

Enabling additional network tags on static and master nodes

This parameter can be set either in an existing cluster or when creating one. In both cases, additional network tags are declared in the `GCPClusterConfiguration`:

- for master nodes, in the `additionalNetworkTags` field of the `masterNodeGroup` section;
- for static nodes, in the `additionalNetworkTags` field of the `nodeGroups` subsection that corresponds to the target nodeGroup.

The `additionalNetworkTags` field contains an array of strings with network tags names.

Enabling additional network tags on ephemeral nodes

You have to set the `additionalNetworkTags` parameter for all `GCPInstanceClass` that require additional network tags.

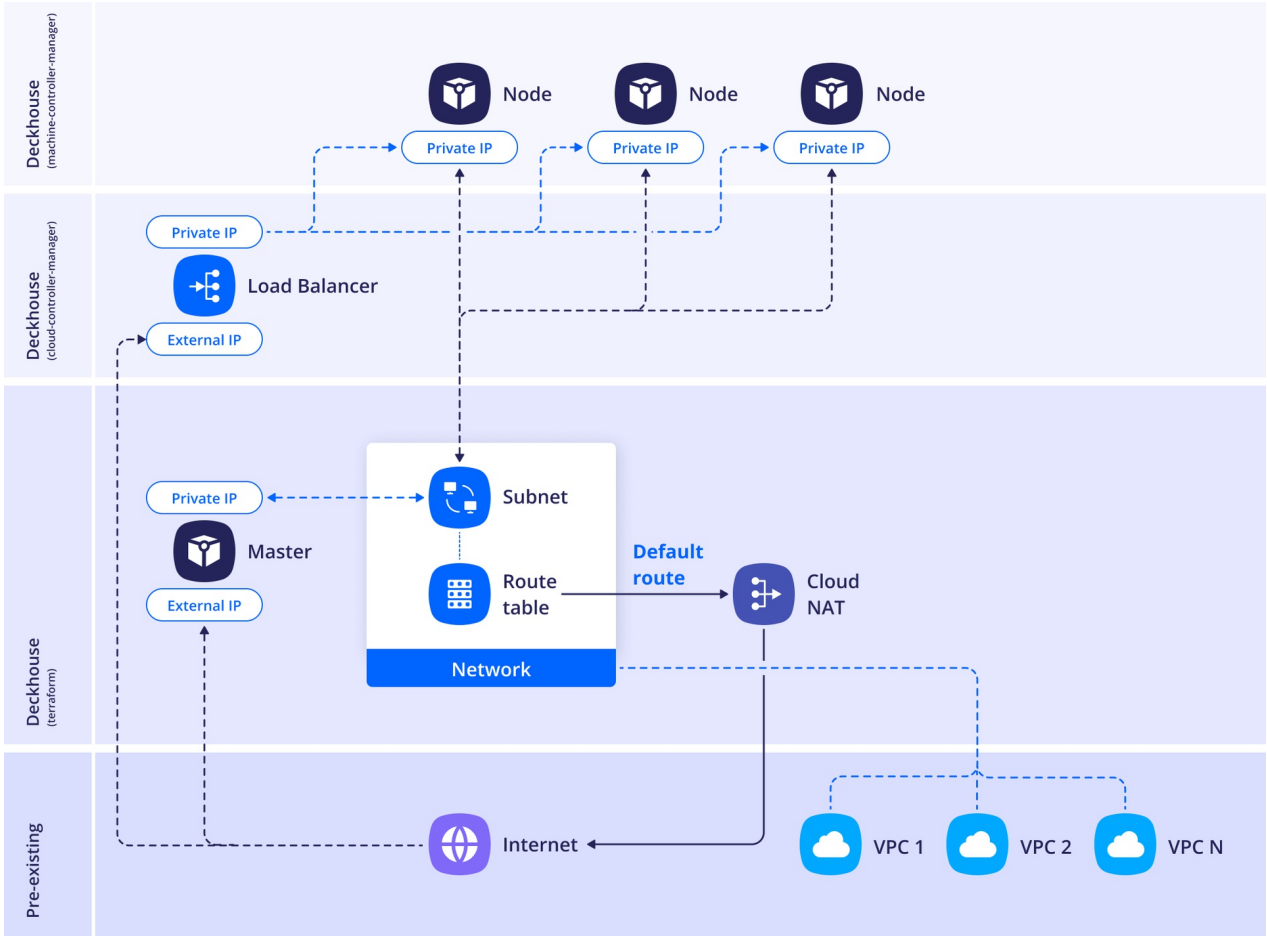
Layouts

> **i** The module lifecycle stage: General Availability

Two layouts are supported. Below is more information about each of them.

Standard

- A separate VPC with Cloud NAT is created for the cluster.
- Nodes in the cluster do not have public IP addresses.
- Public IP addresses can be allocated to master and static nodes:
  - In this case, one-to-one NAT is used to translate the public IP address to the node's IP address (note that CloudNAT is not used in such a case).
- If the master does not have a public IP, then an additional instance with a public IP (aka bastion host) is required for installation tasks and accessing the cluster.
- Peering can also be configured between the cluster VPC and other VPCs.

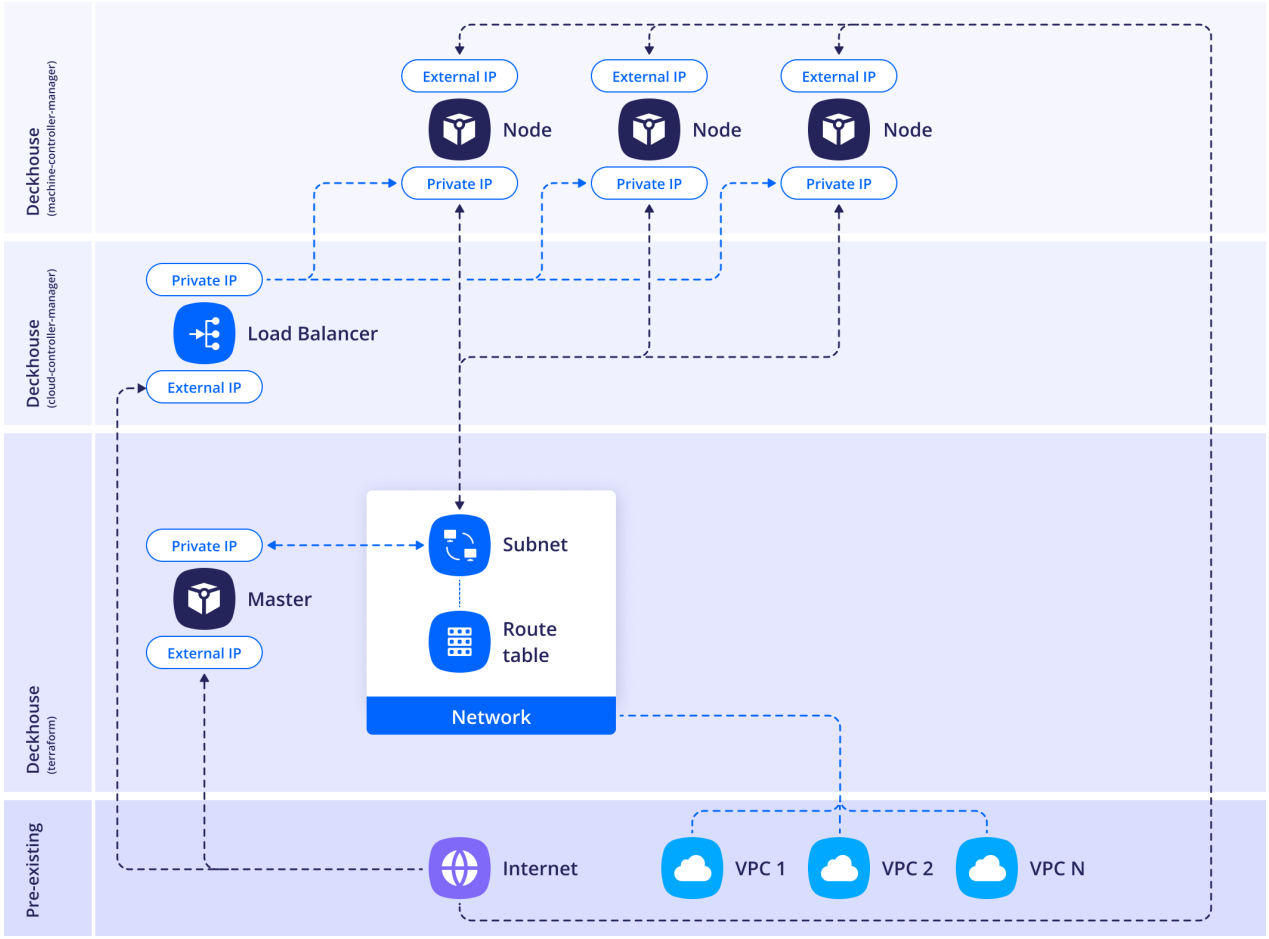


Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: Standard
standard:
  # Optional, compute address names from this list are used as addresses for Cloud NAT.
  cloudNATAddresses:
  - example-address-1
  - example-address-2
subnetworkCIDR: 10.0.0.0/24       # Required.
peeredVPCs:
# Optional, list of GCP VPC Networks with which Kubernetes VPC Network will be peered.
- default
sshKey: "<SSH_PUBLIC_KEY>"  # Required.
labels:
  kube: example
masterNodeGroup:
  replicas: 1
  zones:                       # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128 # Required.
    diskSizeGb: 20               # Optional, local disk is used if not specified.
    disableExternalIP: false       # Optional, by default master has externalIP.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:            # Optional.
      kube-node: master
nodeGroups:
- name: static
  replicas: 1
  zones:                       # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128 # Required.
    diskSizeGb: 20               # Optional, local disk is used if not specified.
    disableExternalIP: true        # Optional, by default nodes do not have externalIP.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:            # Optional.
      kube-node: static
provider:
  region: europe-west4           # Required.
  serviceAccountJSON: |          # Required.
    {
      "type": "service_account",
      "project_id": "sandbox",
      "private_key_id": "98sdcj5e8c7asd98j4j3n9csakn",
      "private_key": "-----BEGIN PRIVATE KEY-----",
      "client_id": "342975658279248",
      "auth_uri": "https://accounts.google.com/o/oauth2/auth",
      "token_uri": "https://oauth2.googleapis.com/token",
      "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
      "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/k8s-
test%40sandbox.iam.gserviceaccount.com"
    }
```

WithoutNAT

- A dedicated VPC is created for the cluster; all cluster nodes have public IP addresses.
- Peering can be configured between the cluster VPC and other VPCs.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: GCPClusterConfiguration
layout: WithoutNAT
subnetworkCIDR: 10.0.0.0/24       # Required.
# Optional, list of GCP VPC Networks with which Kubernetes VPC Network will be peered.
peeredVPCs:
- default
labels:
  kube: example
masterNodeGroup:
  replicas: 1
  zones:                        # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128  # Required.
    diskSizeGb: 20               # Optional, local disk is used if not specified.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:             # Optional.
      kube-node: master
nodeGroups:
- name: static
  replicas: 1
  zones:                        # Optional.
  - europe-west4-b
  instanceClass:
    machineType: n1-standard-4     # Required.
    image: projects/ubuntu-os-cloud/global/images/ubuntu-2404-noble-amd64-v20260128  # Required.
    diskSizeGb: 20               # Optional, local disk is used if not specified.
    additionalNetworkTags:         # Optional.
    - tag1
    additionalLabels:             # Optional.
      kube-node: static
provider:
  region: europe-west4            # Required.
  serviceAccountJSON: |           # Required.
    {
      "type": "service_account",
      "project_id": "sandbox",
      "private_key_id": "98sdcj5e8c7asd98j4j3n9csakn",
      "private_key": "-----BEGIN PRIVATE KEY-----",
      "client_id": "342975658279248",
      "auth_uri": "https://accounts.google.com/o/oauth2/auth",
      "token_uri": "https://oauth2.googleapis.com/token",
      "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
      "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/k8s-
test%40sandbox.iam.gserviceaccount.com"
    }
```

Cloud provider — Huawei Cloud

> **i** The module lifecycle stage: General Availability

The `cloud-provider-huaweicloud` module integrates Deckhouse Kubernetes Platform with Huawei Cloud. It allows the `node-manager` module to use Huawei Cloud resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-huaweicloud` module:

- Managing Huawei Cloud resources via `cloud-controller-manager`:
  - updates instance and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in the cloud;
  - creates load balancers (ELB) for Services of the LoadBalancer type.
- Provisioning disks via the EVS CSI driver (`evs.csi.huaweicloud.com`) so that PersistentVolumes can be requested from the cluster.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-huaweicloud`.
- Provisioning CloudEphemeral nodes via Cluster API (CAPI). Virtual machine parameters are set in the HuaweiCloudInstanceClass resource.
- Registering with `node-manager` so that HuaweiCloudInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

Configuration

> **i** The module lifecycle stage: General Availability

The module is automatically enabled for all cloud clusters deployed in Huawei Cloud.

The module has 1 alert.

The module does not have any settings.

Provider configuration

> **i** The module lifecycle stage: General Availability

HuaweiCloudClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in Huawei Cloud.

Used by the cloud provider if a cluster’s control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

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

Example:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- eu-3a
standard:
  externalNetworkName: external-network
provider:
  domainName: "<DOMAIN_NAME>"
  region: eu-3
  accessKey: "<USERNAME>"
  secretKey: "<PASSWORD>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    flavorName: m1.large
    rootDiskSize: 50
    imageName: debian-11-genericcloud-amd64-20220911-1135
  volumeTypeMap:
    eu-1a: fast-eu-1a
    eu-1b: fast-eu-1b
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.large
    imageName: debian-11-genericcloud-amd64-20220911-1135
    rootDiskSize: 50
  volumeTypeMap:
    eu-1a: fast-eu-1a
    eu-1b: fast-eu-1b
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`
- **kind**
  string

  Required value

  Allowed values: `HuaweiCloudClusterConfiguration`
- **layout**
  string

Required value

The layout name.

For details about possible provider layouts, refer to Layouts.

- **masterNodeGroup**

  Required value

  The definition of a NodeGroup for master nodes.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the fields of the HuaweiCloudInstanceClass.

    - **masterNodeGroup.instanceClass.etcdDiskSizeGb**
      integer

      Etcd disk size in gigabytes.

      Default: `10`

      Example:

      ```
      etcdDiskSizeGb: 10
      ```

    - **masterNodeGroup.instanceClass.flavorName**
      string

      Required value

      The flavor of Huawei Cloud servers.

    - **masterNodeGroup.instanceClass.imageName**
      string

      Required value

      The image to use while provisioning Huawei Cloud servers.

      For the list of operating systems and specific versions supported by Deckhouse, refer to Supported Kubernetes and OS versions (take into account the Deckhouse version you use).

    - **masterNodeGroup.instanceClass.rootDiskSize**
      integer

      The size of a root disk in gigabytes.

      This parameter also affects the type of a root disk.

      Default: `50`

      Example:

      ```
      rootDiskSize: 50
      ```

  - **masterNodeGroup.replicas**
    integer

    Required value

    The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

    Allowed values: `1 <= X`

  - **masterNodeGroup.serverGroup**
    object

    The ServerGroup object groups instances together. The instances in the group are placed on the same hypervisor (affinity) or different hypervisors (anti-affinity). This allows you to increase the fault tolerance of the cluster.

    - **masterNodeGroup.serverGroup.policy**
      string

      Required value

      The policy that determines how instances are distributed among hypervisors.

      - `AntiAffinity`: Instances are placed on different hypervisors. This policy is managed by Deckhouse.

      Allowed values: `AntiAffinity`

- **masterNodeGroup.volumeTypeMap**
  object

  Required value

  A dictionary of disk types for storing etcd data and Kubernetes configuration files.

  Format of dictionary elements: `<AVAILABILITY ZONE>: <DISK TYPE>` (see the example).

  If the `rootDiskSize` parameter is specified, the same disk type will be used for the VM's boot drive.

  We recommend using the fastest disks provided by the provider in all cases.

  If the value specified in `replicas` exceeds the number of elements in the dictionary, the master nodes whose number exceeds the length of the dictionary get the values starting from the beginning of the dictionary. For example, if `replicas: 5`, then master-0, master-2, master-4 will have the `fast-eu-1a` disk type, while master-1, master-3 will have the `fast-eu-1b` disk type.

  Example:

  ```
  volumeTypeMap:
    eu-1a: fast-eu-1a
    eu-1b: fast-eu-1b
  ```

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields.

    - **nodeGroups.instanceClass.additionalNetworks**
      array of strings

      Not required value.

      List of network IDs (subnets) for secondary NICs.

    - **nodeGroups.instanceClass.etcdDiskSizeGb**
      integer

      Etcd disk size in gigabytes.

      Default: `10`

      Example:

      ```
      etcdDiskSizeGb: 10
      ```

    - **nodeGroups.instanceClass.flavorName**
      string

      Required value

      The flavor of Huawei Cloud servers.

    - **nodeGroups.instanceClass.imageName**
      string

      Required value

      The image to use while provisioning Huawei Cloud servers.

      For the list of operating systems and specific versions supported by Deckhouse, refer to Supported Kubernetes and OS versions (take into account the Deckhouse version you use).

    - **nodeGroups.instanceClass.mainNetwork**
      string

      Not required value.

      Primary network ID for the VM's main NIC. **By default**, the value from 'HuaweiCloudDiscoverData' is used.

    - **nodeGroups.instanceClass.rootDiskSize**
      integer

      The size of a root disk in gigabytes.

      This parameter also affects the type of a root disk.

      Default: `50`

      Example:

      ```
      rootDiskSize: 50
      ```

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: 'true'
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to all cluster resources, if they support it.

    The same as the `metadata.labels` standard field.

    If you change a label on a running cluster, recreate all virtual machines after the changes come into effect.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object.

    > **i** Available fields: `effect`, `key`, and `values`.

    Example:

    ```
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
    ```

    - **nodeGroups.nodeTemplate.taints.effect**
      string

      Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

    - **nodeGroups.nodeTemplate.taints.key**
      string
    - **nodeGroups.nodeTemplate.taints.value**
      string

- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes to create.

- **nodeGroups.volumeTypeMap**
  object

  A dictionary mapping availability zones to disk types for the group's worker nodes.

  Format of dictionary elements: `<AVAILABILITY ZONE>: <DISK TYPE>` (see the example).

  If the `rootDiskSize` parameter is specified, the same disk type is used for the VM's boot disk.

  We recommend using the fastest disk type offered by the provider whenever possible.

  If `volumeTypeMap` is not specified for a node group, the `masterNodeGroup`'s `volumeTypeMap` is used as a fallback for that group.

  If the replicas value exceeds the number of entries in the dictionary, assignment wraps around from the beginning. For example, if `replicas: 5` for the `front` group and the dictionary lists `fast-eu-1a` first and `fast-eu-1b` second, then front-0, front-2, front-4 will use `fast-eu-1a`, while front-1, front-3 will use `fast-eu-1b`.

  Example:

```
volumeTypeMap:
  eu-1a: fast-eu-1a
  eu-1b: fast-eu-1b
```

- **provider**
  object

  Required value

  Huawei Cloud API connection settings.

  These settings match the ones in the `connection` field of the cloud-provider-huaweicloud module.

  - **provider.accessKey**
    string

    Required value

    The Huawei Cloud access key.

  - **provider.authURL**
    string

    The identity authentication URL.

  - **provider.cloud**
    string

    Required value

    The endpoint of the cloud provider.

  - **provider.domainName**
    string

    The account name of IAM to scope to.

  - **provider.enterpriseProjectID**
    string

    The enterprise project ID.

  - **provider.insecure**
    boolean

    Trust self-signed SSL certificates.

  - **provider.projectID**
    string

    The project ID.

  - **provider.region**
    string

    Required value

    The Huawei Cloud region where the cluster will be deployed.

  - **provider.secretKey**
    string

    Required value

    The Huawei Cloud secret key.

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **standard**
  object

  Settings for the `Standard` layout.

  - **standard.enableEIP**
    boolean

    Enable Elastic IP for the master nodes.

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  - **standard.internalNetworkCIDR**
    string

    Required value

    Routing for the internal cluster node network.

Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **standard.internalNetworkDNSServers**
  array of strings

  A list of addresses of the recursive DNSs for the internal cluster node network.

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **standard.internalNetworkSecurity**
  boolean

  Defines whether SecurityGroups must be configured for the internal network ports.

  Default: `true`

- **vpcPeering**
  object

  Settings for the `VpcPeering` layout.

  - **vpcPeering.internalNetworkCIDR**
    string

    Required value

    Routing for the internal cluster node network.

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  - **vpcPeering.internalNetworkDNSServers**
    array of strings

    A list of addresses of the recursive DNSs for the internal cluster node network.

    - **Element of the array**
      string

      Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

  - **vpcPeering.internalNetworkSecurity**
    boolean

    Defines whether SecurityGroups must be configured for the internal network ports.

    Default: `true`

  - **vpcPeering.subnet**
    string

    The name of the subnet in the VPC.

- **zones**
  array of strings

  Not required value.

  The globally restricted set of zones that this cloud provider works with.

Preparing environment

> **i** The module lifecycle stage: General Availability

> ⚠ To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:
>
> - Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
> - `CgroupsV2` support.
> - Systemd version `244` or newer.
> - `erofs` kernel module support.
>
> For more information, see the documentation.

⚠ The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Deckhouse Kubernetes Platform components interact with Huawei Cloud resources through the Huawei Cloud API. To configure this connection, you need to create a user in the Huawei Cloud IAM service and provide it with the necessary permissions.

Configuring IAM via the web interface

To configure IAM via the web interface, first create a new user group and assign the necessary permissions. Follow these steps:

1. Go to the “Identity and Access Management (IAM)” section.
2. Open the “User Groups” page and click “Create User Group”.
3. In the “Name” field, enter the group name (e.g., `deckhouse`).
4. Click “OK” to create the group.
5. Select the newly created group from the list.
6. On the “Permissions” tab, click “Authorize”.
7. Assign the following policies: “ECS Admin”, “VPC Administrator”, “NAT Admin”, “ELB FullAccess”, “DEW KeypairFullAccess”.
8. Click “Next”, then “OK”, and complete the setup by clicking “Finish”.

Then add a new user. Follow these steps:

1. Go to the “Users” page in the IAM section and click “Create User”.
2. In the “Username” field, enter the username (e.g., `deckhouse`).
3. Set “Access type” to “Programmatic access” and make sure “Management console access” is disabled.
4. Select “Access key” as the “Credential Type”.
5. Click “Next”.
6. Select the previously created user group.
7. Click “Create” to complete the user creation process.
8. Click “OK” to download the `Access Key ID` and `Secret Access Key`. Make sure to save these credentials in a secure location, as they will be needed to access the API.

JSON policies

Below are the contents of the policies in JSON format:

ECS Admin policy

```
{
"Version": "1.1",
"Statement": [
{
    "Action": [
    "ecs:*:*",
    "evs:*:get",
    "evs:*:list",
    "evs:volumes:create",
    "evs:volumes:delete",
    "evs:volumes:attach",
    "evs:volumes:detach",
    "evs:volumes:manage",
    "evs:volumes:update",
    "evs:volumes:use",
    "evs:volumes:uploadImage",
    "evs:snapshots:create",
    "vpc:*:get",
    "vpc:*:list",
    "vpc:networks:create",
    "vpc:networks:update",
    "vpc:subnets:update",
    "vpc:subnets:create",
    "vpc:ports:*",
    "vpc:routers:get",
    "vpc:routers:update",
    "vpc:securityGroups:*",
    "vpc:securityGroupRules:*",
    "vpc:floatingIps:*",
    "vpc:publicIps:*",
    "ims:images:create",
    "ims:images:delete",
    "ims:images:get",
    "ims:images:list",
    "ims:images:update",
    "ims:images:upload"
    ],
    "Effect": "Allow"
}
]
}
```

VPC Administrator policy

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "vpc:vpcs:*",
                "vpc:routers:*",
                "vpc:networks:*",
                "vpc:subnets:*",
                "vpc:ports:*",
                "vpc:privateIps:*",
                "vpc:peerings:*",
                "vpc:routes:*",
                "vpc:lbaas:*",
                "vpc:vpns:*",
                "ecs:*:get",
                "ecs:*:list",
                "elb:*:get",
                "elb:*:list"
            ],
            "Effect": "Allow"
        }
    ]
}
```

NAT Admin policy

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "nat:*:*",
                "vpc:*:*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

DEW KeypairFullAccess policy

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "kps:domainKeypairs:*",
                "ecs:serverKeypairs:*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

ELB FullAccess policy

```
{
  "Version": "1.1",
  "Statement": [
      {
          "Action": [
              "elb:*:*",
              "vpc:*:get*",
              "vpc:*:list*",
              "ecs:*:get*",
              "ecs:*:list*"
          ],
          "Effect": "Allow"
      }
  ]
}
```

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

HuaweiCloudInstanceClass

Scope: Cluster
Version: v1

Huawei Cloud server group parameters used by `machine-controller-manager` (the node-manager module).

The CloudInstanceClass resource of the `node-manager` module refers to this resource.

Example:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudInstanceClass
metadata:
  name: worker
spec:
  imageName: alt-p11
  flavorName: s7n.xlarge.2
  rootDiskSize: 50
  rootDiskType: SSD
```

- **spec**
  object

  Required value

  - **spec.additionalNetworks**
    array of strings

    Not required value.

    List of networks IDs (subnets) for secondary NICs.

  - **spec.flavorName**
    string

    Required value

    Flavor of Huawei Cloud servers.

    To get a list of all available flavors, run the following command: `cloud ECS ListFlavors`.

    For all non-master nodes, it’s recommended that you use a flavor that includes a local disk. If the cloud provider supports local disks, they’re usually faster and cheaper. The disadvantage of using such a flavor is the inability to migrate nodes between hypervisors.

    Example:

    ```
    flavorName: m1.medium
    ```

  - **spec.imageName**
    string

    Not required value.

    Image to use while provisioning HuaweiCloud servers.

    Use this command to get a list of available images: `cloud IMS ListImages`.

    For the list of operating systems and specific versions supported by Deckhouse, refer to Supported Kubernetes and OS versions (take into account the Deckhouse version you use).

    By default, either the `HuaweiCloudDiscoveryData` value or `instances.imageName` is used.

    Example:

    ```
    imageName: ubuntu-18-04-cloud-amd64
    ```

  - **spec.mainNetwork**
    string

    Not required value.

    ID of the primary network for the VM’s main NIC.

    By default, the value from ‘HuaweiCloudDiscoverData’ is used.

  - **spec.rootDiskSize**
    integer

    Not required value.

    The size of a root disk in gigabytes.

    This parameter also affects the type of a root disk.

    Example:

    ```
    rootDiskSize: 20
    ```

  - **spec.rootDiskType**
    string

    Not required value.

    The type of a root disk.

    The parameter is used to specify the type of a volume that will be used for the root disk.

By default, the value from `HuaweiCloudDiscoveryData` is used.

Example:

```
rootDiskType: GPSSD
```

- **spec.securityGroups**
  array of strings

  Not required value.

  Security groups that will be applied to VM’s network ports in addition to security groups set in a cloud provider configuration.

  These security groups allow you to set firewall rules for provisioned instances.

  The `SecurityGroups` group may not be supported by the cloud provider.

  Example:

  ```
  securityGroups:
  - security-group-1
  - security-group-2
  ```

- **spec.serverGroupID**
  string

  Not required value.

  The server group to assign the machine to.

  Example:

  ```
  serverGroupID: server-group-1
  ```

- **spec.subnets**
  Deprecated
  array of strings

  Not required value.

  > **i** Use the additionalNetworks field instead.

  Paths to networks that the secondary VirtualMachines NICs will connect to.

  Example:

  ```
  subnets:
  - BGP-network-VLAN-3894
  - External-VLAN-3699
  ```

- **spec.vipAddress**
  string

  Not required value.

  If filled, enables Virtual IP for all nodes in this instance class.

  `Auto` or IPv4 address are expected.

  Pattern: `^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$|^Auto$`

  Example:

  ```
  vipAddress: Auto
  ```

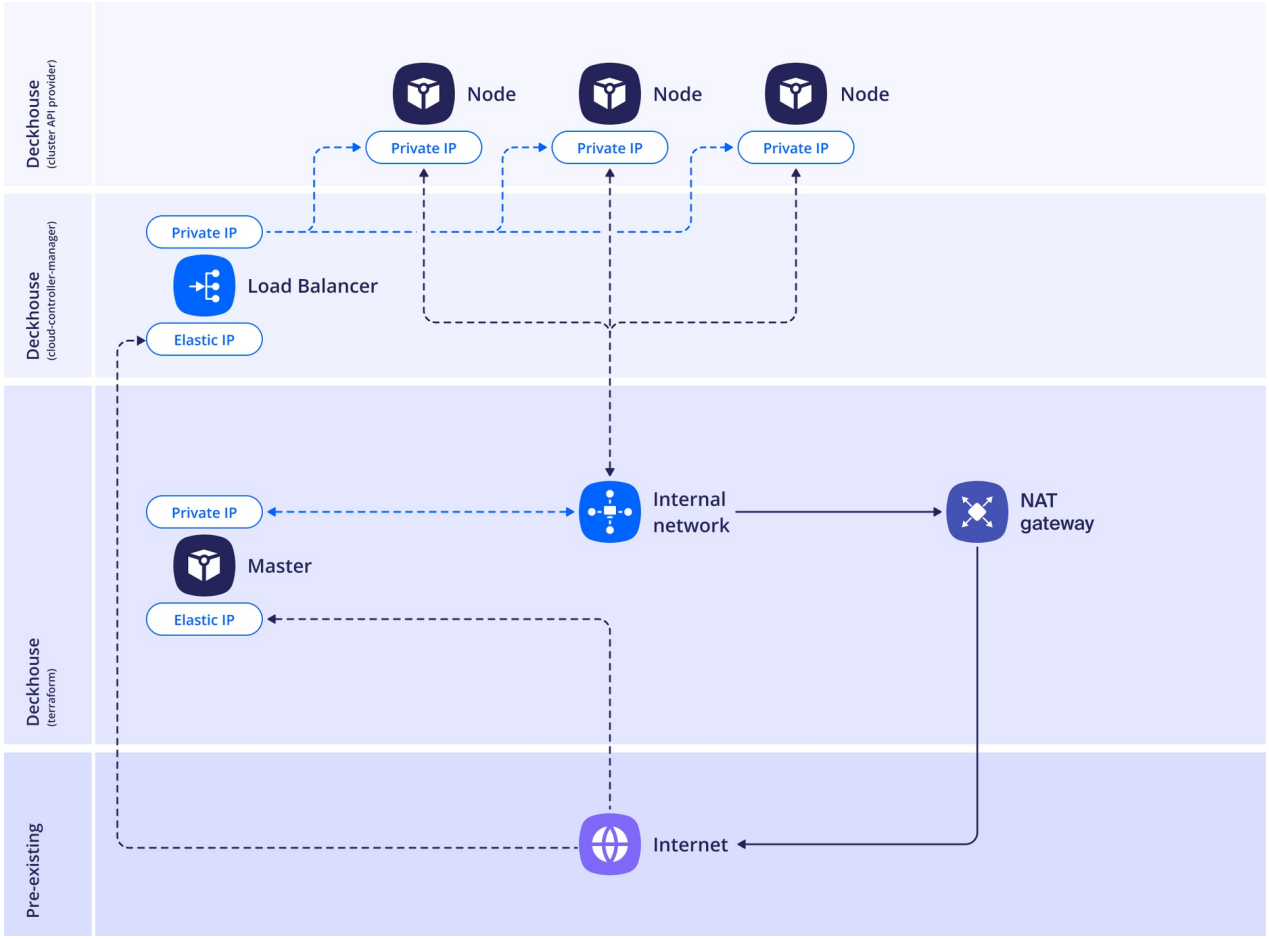
- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Cloud provider - HuaweiCloud: Layouts

> **i** The module lifecycle stage: General Availability

Standard

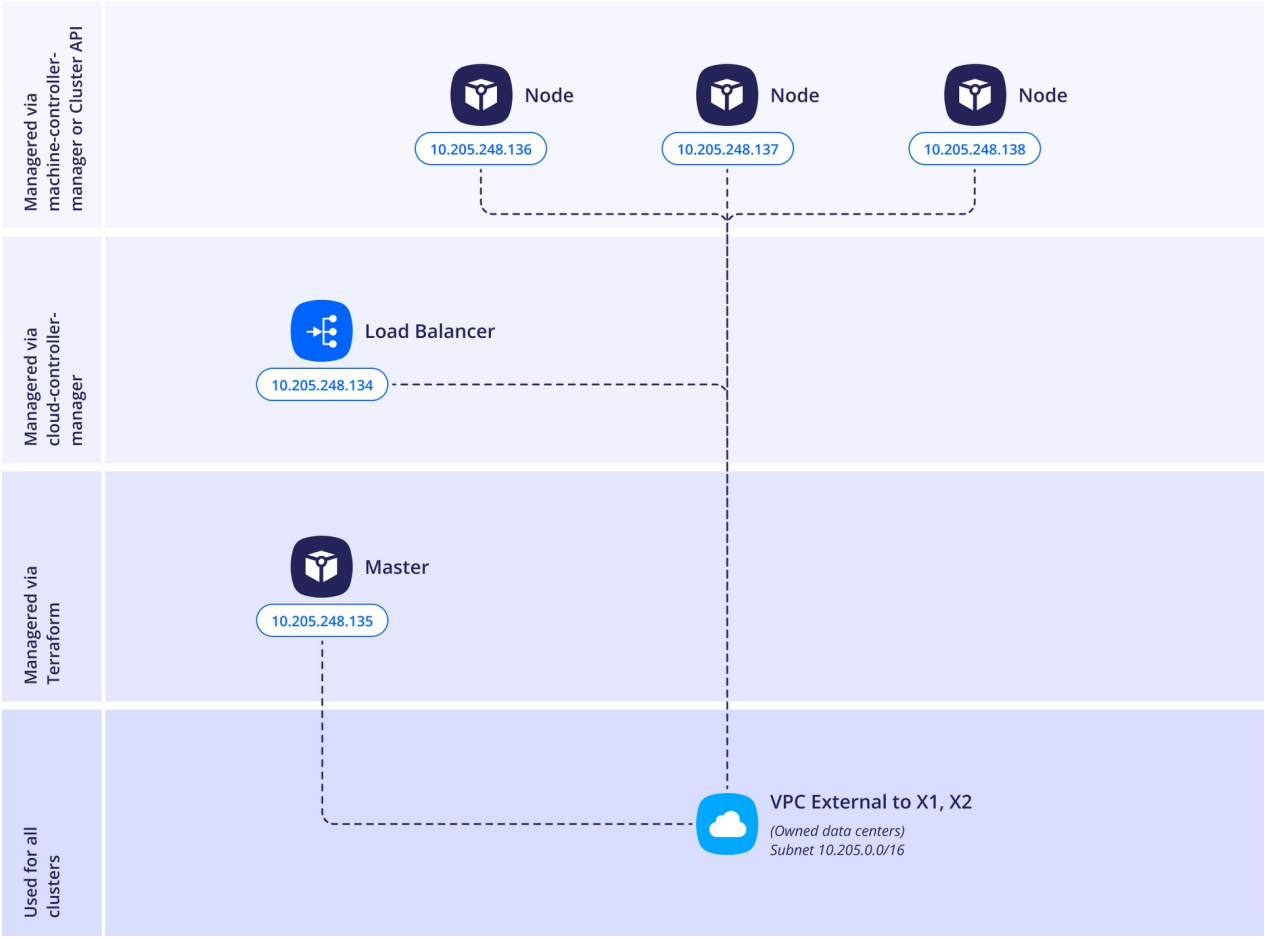
- An internal cluster network is created with a gateway to the public network.
- The elastic IP can be assigned to the master node.
- Nodes managed by the Cluster API do not have public IP addresses.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
standard:
  internalNetworkDNSServers:
    - 8.8.8.8
  internalNetworkCIDR: 192.168.199.0/24
  internalNetworkSecurity: true
  enableEIP: true
provider:
  cloud: hc.sbercloud.ru
  region: ru-moscow-1
  accessKey: "<Access key>"
  secretKey: "<Secret key>"
  projectID: "<Project ID>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    imageName: alt-p11
    flavorName: s7n.xlarge.2
    rootDiskSize: 50
  serverGroup:
    policy: AntiAffinity
  volumeTypeMap:
    ru-moscow-1a: SSD
```

VpcPeering



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: HuaweiCloudClusterConfiguration
layout: VpcPeering
sshPublicKey: "<SSH_PUBLIC_KEY>"
vpcPeering:
  internalNetworkDNSServers:
    - 8.8.8.8
  internalNetworkCIDR: 10.221.128.0/24
  internalNetworkSecurity: true
  subnet: subnet-43b4
```

Cloud provider — OpenStack

> **i** The module lifecycle stage: General Availability

The `cloud-provider-openstack` module integrates Deckhouse Kubernetes Platform with OpenStack-based clouds. It allows the `node-manager` module to use OpenStack resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-openstack` module:

- Managing OpenStack resources via `cloud-controller-manager`:
  - updates OpenStack server and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in OpenStack;
  - creates load balancers (Octavia) for Services of the LoadBalancer type.
- Provisioning block disks via the Cinder CSI driver (`cinder.csi.openstack.org`). Manila (filesystem) is not supported. The Cinder CSI driver supports re-authentication in OpenStack with service catalog refresh, which improves resilience for long-running pods with volumes.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-openstack`.
- Provisioning CloudEphemeral nodes via Machine Controller Manager (MCM) or Cluster API (CAPI). Virtual machine parameters are set in the OpenStackInstanceClass resource.
- Registering with `node-manager` so that OpenStackInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used. The network mode depends on the `podNetworkMode` parameter.

⚠ The module is migrating CloudEphemeral node management from Machine Controller Manager (MCM) to Cluster API (CAPI). Existing NodeGroups continue to use MCM, while new ones are created with CAPI by default. For migrating existing groups, see How to migrate node groups to Cluster API (CAPI).

Configuration

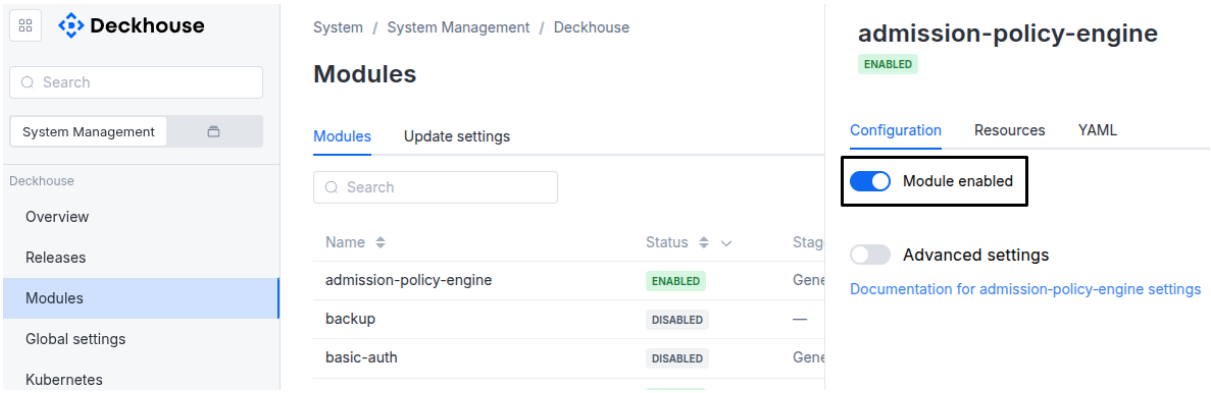
ℹ The module lifecycle stage: General Availability

The module is automatically enabled for all cloud clusters deployed in OpenStack.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-openstack` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cloud-provider-openstack
```

- **Using ModuleConfig `cloud-provider-openstack`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-openstack` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-openstack`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  enabled: true
```

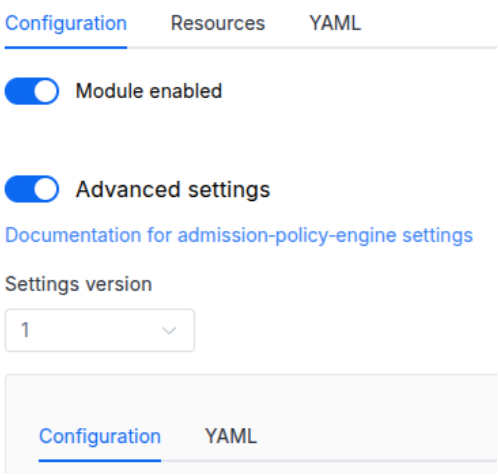
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-openstack` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `cloud-provider-openstack` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-openstack`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-openstack` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-openstack
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cloud-provider-openstack` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cloud-provider-openstack`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-openstack
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

You can configure the number and parameters of ordering machines in the cloud via the `NodeGroup` custom resource of the node-manager module. Also, in this custom resource, you can specify the instance class’s name for the above group of nodes (the `cloudInstances.ClassReference` NodeGroup parameter). In the case of the OpenStack-based cloud provider, the instance class is the `OpenStackInstanceClass` custom resource that stores specific parameters of the machines.

> **i** An OpenStack cluster is deployed in a single region (`provider.region` in OpenStackClusterConfiguration). Zones from other regions cannot be used. See How do I create NodeGroups in availability zones? .

The module settings are set automatically based on the placement strategy chosen. In most cases, you do not have to configure the module manually.

If you need to configure a module because, say, you have a bare metal cluster and you need to enable additional instances from vSphere, then refer to the How to configure a Hybrid cluster in vSphere section.

> **i** **Note!** If the parameters provided below are changed, the **existing `Machines` are NOT redeployed** (new `Machines` will be created with the updated parameters). Redeployment is only performed when `NodeGroup` and `OpenStackInstanceClass` parameters are changed. You can learn more in the node-manager module’s documentation. To authenticate using the `user-authn` module, you need to create a new `Generic` application in the project’s Crowd.

List of required OpenStack services

A list of OpenStack services required for Deckhouse Kubernetes Platform to work in OpenStack:

Service	API Version
Identity (Keystone)	v3
Compute (Nova)	v2
Network (Neutron)	v2
Block Storage (Cinder)	v3
Load Balancing (Octavia) *	v2

* If you need to order a Load Balancer.

To ensure proper integration and functionality, the OpenStack user associated with the Deckhouse Kubernetes Platform should be granted the “member” role. This role assignment is essential to provide the necessary permissions for interacting with the listed services and managing resources within the OpenStack environment.

Parameters

Schema version: 1

- **settings**
  object
  - **settings.additionalExternalNetworkNames**
    array of strings

    Specifies additional networks that can be connected to the VM. `cloud-controller-manager` uses them to insert `ExternalIP` to `.status.addresses` field in the Node API object.

    If you have instances in the cluster that use External Networks (other than those set out in the placement strategy), you must pass them via the `additionalExternalNetworkNames` parameter.

    Example:

    ```
    additionalExternalNetworkNames:
    - some-bgp-network
    ```

  - **settings.connection**
    object

    This section contains parameters required to connect to the cloud provider’s API.

    - **settings.connection.authURL**
      string

      An OpenStack Identity API URL.

    - **settings.connection.caCert**
      string

      Specify the CA x509 certificate used for signing if the OpenStack API has a self-signed certificate; The certificate must have a PEM format.

    - **settings.connection.domainName**
      string

      The domain name.

    - **settings.connection.password**
      string

      The user’s password.

    - **settings.connection.region**
      string

      The OpenStack region where the cluster will be deployed.

    - **settings.connection.tenantID**
      string

      The project ID. Cannot be used together with `tenantName`.

    - **settings.connection.tenantName**
      string

      The project name.

      Cannot be used together with `tenantID`.

    - **settings.connection.username**
      string

      The name of the user that has full project privileges.

  - **settings.csiDriver**
    object

    Parameters for configuring the Cinder CSI Driver.

- **settings.csiDriver.fsGroupPolicy**
  string

  Defines if the underlying volume supports changing ownership and permission of the volume before being mounted.

  - `ReadWriteOnceWithFSType`: Apply fsGroup only for ReadWriteOnce volumes (Kubernetes default behavior).
  - `File`: Always apply fsGroup. Use this for RWX volumes with Ceph backend.
  - `None`: Never change permissions.

  Default: `ReadWriteOnceWithFSType`

  Allowed values: `ReadWriteOnceWithFSType`, `File`, `None`

- **settings.externalNetworkNames**
  array of strings

  Additional networks that are connected to the VM. cloud-controller-manager uses them to insert ExternalIPs into `.status.addresses` in the Node API object.

  Example:

  ```
  externalNetworkNames:
  - KUBE-3
  - devops-internal
  ```

- **settings.ignoreVolumeMicroversion**
  boolean

  Setting for backwards compatibility. Enable if the cloud OpenStack version is less than 3.34 and you get error “Version 3.34 is not supported by the API. Minimum is 3.0 and maximum is 3.x” when ordering a PV. This will disable volumes online resize, but will restore ability to order new PVs. (original PR https://github.com/kubernetes/cloud-provider-openstack/pull/1986/)

  Default: `false`

- **settings.instances**
  object

  Instance parameters that are used when creating virtual machines.

  - **settings.instances.additionalNetworks**
    array of strings

    A list of networks to connect to the instance.

  - **settings.instances.imageName**
    string

    The name of the image.

  - **settings.instances.mainNetwork**
    string

    The path to the network that will serve as the primary network (the default gateway) for connecting to the VM.

  - **settings.instances.securityGroups**
    array of strings

    A list of securityGroups to assign to the provisioned instances. Defines firewall rules for the provisioned instances.

  - **settings.instances.sshKeyPairName**
    string

    The name of the OpenStack `keypair` resource; it is used for provisioning instances.

- **settings.internalNetworkNames**
  array of strings

  Additional networks that are connected to the VM. cloud-controller-manager uses them to insert InternalIPs into `.status.addresses` in the Node API object.

  Example:

  ```
  internalNetworkNames:
  - KUBE-3
  - devops-internal
  ```

- **settings.loadBalancer**
  object

  Load Balancer parameters.

  - **settings.loadBalancer.disabled**
    boolean

Set to `true` to disable LoadBalancer support if the OpenStack cloud doesn't have Octavia (LBaaS) service available.

Default: `false`

- **settings.loadBalancer.floatingNetworkID**
  string

  An ID of the external network for floating IPs.

- **settings.loadBalancer.subnetID**
  string

  An ID of the Neutron subnet to create the load balancer virtual IP in.

- **settings.podNetworkMode**
  string

  Sets the traffic mode for the network that the pods use to communicate with each other (usually, it is an internal network; however, there can be exceptions):

  - `DirectRouting` — means that there is a direct routing between the nodes.
  - `DirectRoutingWithPortSecurityEnabled` — direct routing is enabled between the nodes, but only if the range of addresses of the internal network is explicitly allowed in OpenStack for Ports.
    - **Caution!** Make sure that the `username` can edit AllowedAddressPairs on Ports connected to the `internalNetworkName` network. Generally, an OpenStack user doesn't have such a privilege if the network has the `shared` flag set.
  - `VXLAN` — direct routing between the nodes isn't available; VXLAN should be used.

  Allowed values: `DirectRouting`, `DirectRoutingWithPortSecurityEnabled`, `VXLAN`

- **settings.storageClass**
  object

  The module automatically creates StorageClasses that are available in OpenStack.

  - **settings.storageClass.default**
    Deprecated
    string

    The name of StorageClass that will be used in the cluster by default.

    If the parameter is omitted, the default StorageClass is either:

    - an arbitrary StorageClass present in the cluster that has the default annotation;
    - the first (in lexicographic order) StorageClass created by the module (in accordance with the order in OpenStack).

    > **Parameter is deprecated**. Instead, use the global parameter global.defaultClusterStorageClass.

    Example:

    ```
    default: ceph-ssd
    ```

  - **settings.storageClass.exclude**
    array of strings

    A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

    Example:

    ```
    exclude:
    - ".*-hdd"
    - iscsi-fast
    ```

  - **settings.storageClass.topologyEnabled**
    boolean

    This feature enables driver to consider the topology constraints while creating the volume. It is used only during volume provisioning, existing PersistentVolumes are not affected.

    > **Warning.** If it is set to `false` all-new PersistentVolumes are provisioned without topology constraints.

    Default: `true`

- **settings.tags**
  object

  A dictionary of tags that will be available on all provisioned instances.

- **settings.zones**
  array of strings

  The default list of zones for provisioning instances. Can be redefined for each NodeGroup individually.

Provider configuration

> **i** The module lifecycle stage: General Availability

OpenStackClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in OpenStack.

Used by the cloud provider if a cluster’s control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

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

> **i** After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- eu-3a
- eu-3b
standard:
  internalNetworkDNSServers:
  - 8.8.8.8
  internalNetworkCIDR: 192.168.195.0/24
  internalNetworkSecurity: false
  externalNetworkName: external-network
provider:
  authURL: "<AUTH_URL>"
  domainName: "<DOMAIN_NAME>"
  tenantID: "<TENANT_ID>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  region: eu-3
masterNodeGroup:
  replicas: 1
  instanceClass:
    rootDiskSize: 50
    flavorName: m1.large
    imageName: debian-11-genericcloud-amd64-20220911-1135
  volumeTypeMap:
    eu-3a: fast.eu-3a
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.large
    imageName: debian-11-genericcloud-amd64-20220911-1135
    rootDiskSize: 50
    configDrive: false
    floatingIPPools:
    - public
    - shared
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - eu-3a
  - eu-3b
```

- **apiVersion**
  string

  Required value

Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **kind**
  string

  Required value

  Allowed values: `OpenStackClusterConfiguration`

- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

- **masterNodeGroup**

  Required value

  The definition of the master's NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the fields of the OpenStackInstanceClass.

    - **masterNodeGroup.instanceClass.additionalSecurityGroups**
      array of strings

      Security groups that will be applied to VM's network ports. Default group should also be added to this list.

      They allow you to set firewall rules for provisioned instances.

      The `SecurityGroups` may not be supported by the cloud provider.

    - **masterNodeGroup.instanceClass.additionalTags**
      object

      The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

      Example:

      ```
      additionalTags:
        project: cms-production
        severity: critical
      ```

    - **masterNodeGroup.instanceClass.etcdDiskSizeGb**
      integer

      Etcd disk size in gigabytes.

      Default: `10`

      Example:

      ```
      etcdDiskSizeGb: 10
      ```

    - **masterNodeGroup.instanceClass.flavorName**
      string

      Required value

      Flavor of OpenStack servers.

      To get a list of all available flavors, run the command: `openstack flavor list`.

      For all non-master nodes it is advisable to use flavor's with a local disk. If cloud provider supports local disks they are usually faster and cheaper. The disadvantage of using such flavors is the inability to migrate nodes between hypervisors.

      Flavor create example: `openstack flavor create c4m8d50 --ram 8192 --disk 50 --vcpus 4`

    - **masterNodeGroup.instanceClass.imageName**
      string

      Required value

      Image to use while provisioning OpenStack servers.

      Use this command to get a list of available images: `openstack image list`.

The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

- **masterNodeGroup.instanceClass.rootDiskSize**
  integer

  The size of a root disk (in gigabytes).

  This parameter also has influence on type of volume that will be used for root disk; the “How to use rootDiskSize and when it is preferred” section describes how to use it.

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.serverGroup**
  object

  Object groups instances together. The instances in the group are placed on the same hypervisor (affinity) or different hypervisors (anti-affinity). This allows you to increase the fault tolerance of the cluster.

  - **masterNodeGroup.serverGroup.manuallyManaged**
    object
    - **masterNodeGroup.serverGroup.manuallyManaged.id**
      string

      Required value

      The id of the existing `ServerGroup` object managed outside of Deckhouse.
  - **masterNodeGroup.serverGroup.policy**
    string

    Required value

    The policy that determines how instances are distributed among hypervisors.

    - `AntiAffinity` — instances are placed on different hypervisors. This policy is managed by Deckhouse.
    - `ManuallyManaged` — instances are placed on different hypervisors. This policy is managed outside of Deckhouse.

    Allowed values: `AntiAffinity`, `ManuallyManaged`

- **masterNodeGroup.volumeTypeMap**
  object

  Required value

  A dictionary of disk types for storing etcd data and Kubernetes configuration files.

  Format of dictionary elements: `<AVAILABILITY ZONE>: <VOLUME TYPE>` (see the example).

  If the `rootDiskSize` parameter is specified, the same disk type will be used for the VM’s boot drive.

  We recommend using the fastest disks provided by the provider in all cases.

  If the value specified in `replicas` exceeds the number of elements in the dictionary, the master nodes whose number exceeds the length of the dictionary get the values starting from the beginning of the dictionary. For example, if `replicas: 5`, then master-0, master-2, master-4 will have the `fast-eu-1a` disk type, while master-1, master-3 will have the `fast-eu-1b` disk type.

  Master nodes will be created in zones specified at this param following by alphabetical order. Based on the example below, `master-0` will be created in zone `eu-1a`, `master-1` will be created in zone `eu-1b`, and `master-2` will be created in zone `eu-1a`.

  Useful commands:

  - `openstack availability zone list` — get list of availability zones.
  - `openstack volume type list` — get list of volume types.

  Example:

  ```
  volumeTypeMap:
    eu-1a: fast-eu-1a
    eu-1b: fast-eu-1b
  ```

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating CloudPermanent nodes (for example, dedicated front nodes or gateways).

  For CloudEphemeral nodes, use a separate NodeGroup resource and an OpenStackInstanceClass.

  - **nodeGroups.instanceClass**

object

Partial contents of the fields of the OpenStackInstanceClass.

- **nodeGroups.instanceClass.additionalNetworks**
  array of strings

  Paths to networks that VirtualMachines secondary NICs will connect to. To get a list of all available networks, run the command: `openstack network list`.

  Example:

  ```
  additionalNetworks:
  - BGP-network-VLAN-3894
  - External-VLAN-3699
  ```

- **nodeGroups.instanceClass.additionalSecurityGroups**
  array of strings

  Security groups that will be applied to VM’s network ports. Default group should also be added to this list.

  They allow you to set firewall rules for provisioned instances.

  The `SecurityGroups` may not be supported by the cloud provider.

- **nodeGroups.instanceClass.additionalTags**
  object

  The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

  Example:

  ```
  additionalTags:
    project: cms-production
    severity: critical
  ```

- **nodeGroups.instanceClass.configDrive**
  boolean

  Not required value.

  Specifies whether an additional disk containing the bootstrapping configuration will be mounted to the node.

  You must set it if DHCP is disabled in the `mainNetwork`.

  Default: `false`

- **nodeGroups.instanceClass.flavorName**
  string

  Required value

  Flavor of OpenStack servers.

  To get a list of all available flavors, run the command: `openstack flavor list`.

  For all non-master nodes it is advisable to use flavor’s with a local disk. If cloud provider supports local disks they are usually faster and cheaper. The disadvantage of using such flavors is the inability to migrate nodes between hypervisors.

  Flavor create example: `openstack flavor create c4m8d50 --ram 8192 --disk 50 --vcpus 4`

- **nodeGroups.instanceClass.floatingIPPools**
  array of strings

  A list of networks to assign Floating IPs to nodes.

- **nodeGroups.instanceClass.imageName**
  string

  Required value

  Image to use while provisioning OpenStack servers.

  Use this command to get a list of available images: `openstack image list`.

  The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

- **nodeGroups.instanceClass.mainNetwork**
  string

  Required value

  Path to the network that VirtualMachines primary NICs will connect to (default gateway). To get a list of all available networks, run the command: `openstack network list`.

- **nodeGroups.instanceClass.networksWithSecurityDisabled**

array of strings

A list of `mainNetwork` and `additionalNetworks` in which `SecurityGroups` and `AllowedAddressPairs` on ports **cannot be configured**.

- **nodeGroups.instanceClass.rootDiskSize**
  integer

  The size of a root disk (in gigabytes).

  This parameter also has influence on type of volume that will be used for root disk; the “How to use rootDiskSize and when it is preferred” section describes how to use it.

- **nodeGroups.name**
  string

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: 'true'
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to cluster resources.

    The same as the `metadata.labels` standard field.

    Note that you have to re-create all the machines to add new tags if tags were modified in the running cluster.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object.

    > **i** Available fields: `effect`, `key`, and `values`.

    Example:

    ```
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
    ```

    - **nodeGroups.nodeTemplate.taints.effect**
      string

      Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

    - **nodeGroups.nodeTemplate.taints.key**
      string
    - **nodeGroups.nodeTemplate.taints.value**
      string

- **nodeGroups.replicas**
  integer

  The number of nodes to create.

- **nodeGroups.volumeTypeMap**
  object

  A dictionary of disk types for root drive.

  Format of dictionary elements: `<AVAILABILITY ZONE>: <VOLUME TYPE>` (see the example).

  If the value specified in `replicas` exceeds the number of elements in the dictionary, the nodes whose number exceeds the length of the dictionary get the values starting from the beginning of the dictionary. For example, if

`replicas: 5`, then worker-0, worker-2, worker-4 will have the `fast-eu-1a` disk type, while worker-1, worker-3 will have the `fast-eu-1b` disk type.

> **Caution.** Limits by availability zones works only when using the `zones` parameter (`nodeGroups.zones`). If `zones` parameter is not specified, all availability zones will be used.

Useful commands:

- `openstack availability zone list` — get list of availability zones.
- `openstack volume type list` — get list of volume types.

Example:

```
volumeTypeMap:
  eu-1a: fast-eu-1a
  eu-1b: fast-eu-1b
```

- **nodeGroups.zones**
  array of strings

  Not required value.

  A limited set of zones in which nodes can be created.

  The zones must belong to the `provider.region` region. If the `zones` parameter is set, only zones from that parameter can be used.

- **provider**
  object

  Required value

  Contains settings to connect to the OpenStack API.

  These settings are the same as those in the `connection` field of the cloud-provider-openstack module.

  - **provider.authURL**
    string

    An OpenStack Identity API URL.

  - **provider.caCert**
    string

    Specify the CA x509 certificate used for signing if the OpenStack API has a self-signed certificate. Certificate should be passed in PEM format as multiline string.

    Example:

    ```
    caCert: |
      -----BEGIN CERTIFICATE-----
      MIIFyDCCBLCgAwIBAgIQBwDIWH1asdaKNaALUa4NUzANBgkqhkiG9w0BAQsFADBc
      ...
      -----END CERTIFICATE-----
    ```

  - **provider.domainName**
    string

    The domain name.

    `OS_USER_DOMAIN_NAME` variable from the openrc file.

  - **provider.password**
    string

    The user’s password.

  - **provider.region**
    string

    The OpenStack region where the cluster will be deployed.

    The cluster runs in a single region. Master nodes, as well as CloudPermanent and CloudEphemeral nodes, can be created only in zones of this region.

  - **provider.tenantID**
    string

    The project id.

    Cannot be used together with `tenantName`.

  - **provider.tenantName**
    string

    The project name.

Cannot be used together with `tenantID`.

- **provider.username**
  string

  The name of the user that has full project privileges.

- **simple**
  object

  Settings for the `Simple` layout.

  - **simple.externalNetworkDHCP**
    boolean

    This parameter defines if DHCP is enabled in the external network.

    Default: `true`

  - **simple.externalNetworkName**
    string

    Required value

    The name of the network for external connections. To get a list of all available networks, run the command: `openstack network list`.

  - **simple.podNetworkMode**
    string

    Sets the traffic mode for the network that the pods use to communicate with each other (usually, it is an internal network; however, there can be exceptions):

    - `DirectRouting` — nodes are directly routed (SecurityGroups are disabled in this mode).
    - `VXLAN` — direct routing does NOT work between nodes, VXLAN must be used (SecurityGroups are disabled in this mode).

    > **Caution.** After changing this parameter, you need to run `dhctl converge` command for the changes to take effect.

    > **Caution.** All cluster nodes must be rebooted after switching work mode from/to VXLAN.

    Default: `VXLAN`

    Allowed values: `VXLAN`, `DirectRouting`

- **simpleWithInternalNetwork**
  object

  Settings for the `SimpleWithInternalNetwork` layout.

  - **simpleWithInternalNetwork.externalNetworkDHCP**
    boolean

    This parameter defines if DHCP is enabled in the external network.

    Default: `true`

  - **simpleWithInternalNetwork.externalNetworkName**
    string

    The name of the network for external connections. To get a list of all available networks, run the command: `openstack network list`.

  - **simpleWithInternalNetwork.internalSubnetName**
    string

    Required value

    The name of the subnet in which the cluster nodes will run.

  - **simpleWithInternalNetwork.masterWithExternalFloatingIP**
    boolean

    Defines if Floating IP must be assigned to master nodes.

    Default: `true`

  - **simpleWithInternalNetwork.podNetworkMode**
    string

    Sets the traffic mode for the network that the pods use to communicate with each other (usually, it is an internal network; however, there can be exceptions):

    - `DirectRouting` — nodes are directly routed (SecurityGroups are disabled in this mode).
    - `DirectRoutingWithPortSecurityEnabled` — direct routing is enabled between the nodes, but only if the range of addresses of the internal network is explicitly allowed in OpenStack for Ports:
      - **Caution.** Make sure that the `username` can edit `AllowedAddressPairs` on Ports connected to the

`internalNetworkName` network. Usually, an OpenStack user doesn’t have such a privilege if the network has the `shared` flag set.

- `VXLAN` — direct routing does NOT work between nodes, VXLAN must be used (`SecurityGroups` are disabled in this mode).

> **Caution.** After changing this parameter, you need to run `dhctl converge` command for the changes to take effect.

> **Caution.** All cluster nodes must be rebooted after switching work mode from/to VXLAN.

Default: `DirectRoutingWithPortSecurityEnabled`

Allowed values: `VXLAN`, `DirectRouting`, `DirectRoutingWithPortSecurityEnabled`

- **sshAllowList**
  array of strings

  A list of CIDR’s allowed to connect to nodes via SSH.

  By default, `0.0.0.0/0`.

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **standard**
  object

  Settings for the `Standard` layout.

  - **standard.bastion**
    object

    The definition of the bastion instance.

    - **standard.bastion.instanceClass**
      object

      Partial contents of the fields of the OpenStackInstanceClass.

      - **standard.bastion.instanceClass.additionalTags**
        object

        The additional tags to attach to the instance created (in addition to those specified in the cloud provider configuration).

        Example:

        ```
        additionalTags:
          project: cms-production
          severity: critical
        ```

      - **standard.bastion.instanceClass.flavorName**
        string

        Required value

        Flavor of OpenStack servers.

        To get a list of all available flavors, run the command: `openstack flavor list`.

      - **standard.bastion.instanceClass.imageName**
        string

        Required value

        Image to use while provisioning OpenStack servers.

        Use this command to get a list of available images: `openstack image list`.

        The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

      - **standard.bastion.instanceClass.rootDiskSize**
        integer

        The size of a root disk (in gigabytes).

        This parameter also has influence on type of volume that will be used for root disk; the “How to use rootDiskSize and when it is preferred” section describes how to use it.

        Default: `50`

    - **standard.bastion.volumeType**

string

Root disk type.

- **standard.bastion.zone**
  string

  The zone to create an instance for the bastion node.

- **standard.externalNetworkName**
  string

  Required value

  The name of the network for external connections. To get a list of all available networks, run the command: `openstack network list`.

- **standard.internalNetworkCIDR**
  string

  Required value

  Routing for the internal cluster network.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **standard.internalNetworkDNSServers**
  array of strings

  Required value

  A list of addresses of the recursive DNSs of the internal cluster network.

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **standard.internalNetworkSecurity**
  boolean

  Defines whether SecurityGroups and AllowedAddressPairs must be configured for ports of the internal network.

  Default: `true`

- **standardWithNoRouter**
  object

  Settings for the `StandardWithNoRouter` layout.

  - **standardWithNoRouter.externalNetworkDHCP**
    boolean

    This parameter defines if DHCP is enabled in the external network.

    Default: `true`

  - **standardWithNoRouter.externalNetworkName**
    string

    Required value

    The name of the network for external connections. To get a list of all available networks, run the command: `openstack network list`.

  - **standardWithNoRouter.internalNetworkCIDR**
    string

    Required value

    Routing for the internal cluster network.

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  - **standardWithNoRouter.internalNetworkSecurity**
    boolean

    Defines whether SecurityGroups and AllowedAddressPairs must be configured for ports of the internal network.

    Default: `true`

- **tags**
  object

  Not required value.

  A dictionary of tags to create on all resources that support this feature.

  You have to re-create all the machines to add new tags if tags were modified in the running cluster.

- **zones**

array of strings

Not required value.

The globally restricted set of availability zones that this cloud provider works with.

Specify only zones of the `provider.region` region. The cluster is deployed in a single OpenStack region — zones from other regions cannot be used.

To get the list of zones in the region, run: `openstack availability zone list --compute`.

Preparing environment

> **i** The module lifecycle stage: General Availability

> ⚠ To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:
>
> - Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
> - `CgroupsV2` support.
> - Systemd version `244` or newer.
> - `erofs` kernel module support.
>
> For more information, see the documentation.

> ⚠ The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:
>
> - `cloud-config.service`;
> - `cloud-final.service`;
> - `cloud-init.service`.

> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

To manage resources in an OpenStack cloud, Deckhouse connects to the OpenStack API.
The list of OpenStack API services that need to be accessed for deployment is available in the settings section.
The user credentials required to connect to the OpenStack API are located in the openrc file (OpenStack RC file).

You can read the OpenStack documentation to get information about getting and using an openrc file with the standard OpenStack web interface.

The interface for getting an openrc file may differ if you use an OpenStack API of a cloud provider.

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

OpenStackInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of a group of OpenStack servers used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalNetworks**
    array of strings

    Paths to networks that VirtualMachines' secondary NICs will connect to.

    By default: the value from `OpenStackCloudDiscoveryData` is used.

    Example:

```
additionalNetworks:
- BGP-network-VLAN-3894
- External-VLAN-3699
```

- **spec.additionalSecurityGroups**
  array of strings

  Security groups that will be applied to VM’s network ports. Default group should also be added to this list.

  They allow you to set firewall rules for provisioned instances.

  The `SecurityGroups` may not be supported by the cloud provider.

  Example:

  ```
  additionalSecurityGroups:
  - security-group-1
  - security-group-2
  ```

- **spec.additionalTags**
  object

  The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

  Example:

  ```
  additionalTags:
    project: cms-production
    severity: critical
  ```

- **spec.capacity**
  Deprecated
  object

  Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

  Instance capacity for *non-standard* instance types (the `flavorName` parameter).

  Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

  The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`m1.medium`, `m1.large`, etc.) is embedded in Deckhouse.

  - **spec.capacity.cpu**
    integer or string

    Required value

    Node vCPU resources.

    Pattern: `^[0-9]+m?$`

    Example:

    ```
    cpu: 1000m
    ```

  - **spec.capacity.memory**
    integer or string

    Required value

    Node memory resources.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    Example:

    ```
    memory: 1500Mi
    ```

- **spec.flavorName**
  string

  Required value

  Flavor of OpenStack servers.

  Get a list of all available flavors: `openstack flavor list`.

  For all non-master nodes it is advisable to use flavor’s with a local disk. If cloud provider supports local disks they are usually faster and cheaper. The disadvantage of using such flavors is the inability to migrate nodes between hypervisors.

Flavor create example: `openstack flavor create c4m8d50 --ram 8192 --disk 50 --vcpus 4`

Example:

```
flavorName: m1.medium
```

- **spec.imageName**
  string

  Not required value.

  Image to use while provisioning OpenStack servers.

  Use this command to get a list of available images: `openstack image list`.

  The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account the Deckhouse version used).

  By default: either `OpenStackCloudDiscoveryData` value or `instances.imageName` is used.

  Example:

  ```
  imageName: ubuntu-18-04-cloud-amd64
  ```

- **spec.mainNetwork**
  string

  Not required value.

  Path to the network that VirtualMachines' primary NICs will connect to (default gateway).

  By default: the value from `OpenStackCloudDiscoveryData` is used.

  Example:

  ```
  mainNetwork: k8s-msk-178
  ```

- **spec.rootDiskSize**
  integer

  Not required value.

  The size of a root disk (in gigabytes). This parameter also has influence on type of volume that will be used for root disk; the "How to use rootDiskSize and when it is preferred" section describes how to use it.

  Example:

  ```
  rootDiskSize: 20
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Parameters of a group of OpenStack servers used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalNetworks**
    array of strings

    Paths to networks that VirtualMachines' secondary NICs will connect to.

    By default: the value from `OpenStackCloudDiscoveryData` is used.

    Example:

    ```
    additionalNetworks:
    - BGP-network-VLAN-3894
    - External-VLAN-3699
    ```

  - **spec.additionalSecurityGroups**
    array of strings

    Security groups that will be applied to VM's network ports. Default group should also be added to this list.

    They allow you to set firewall rules for provisioned instances.

    The `SecurityGroups` may not be supported by the cloud provider.

    Example:

```
additionalSecurityGroups:
- security-group-1
- security-group-2
```

- **spec.additionalTags**
  object

  The additional tags to attach to the instances created (in addition to those specified in the cloud provider configuration).

  Example:

  ```
  additionalTags:
    project: cms-production
    severity: critical
  ```

- **spec.capacity**
  Deprecated
  object

  Deprecated: the parameter is no longer used. Deckhouse is using parameters from the cloud for passed instance type.

  Instance capacity for *non-standard* instance types (the `flavorName` parameter).

  Cluster-autoscaler uses the parameter for scheduling only when there are no nodes in NodeGroup yet (if `minPerZone` equal to 0). If there are already nodes in the NodeGroup, then cluster-autoscaler uses the actual node capacity (CPU, memory) when planning, and does not use the `capacity` parameter.

  The parameter must be specified only for *non-standard* instance types. Info about the capacity of *standard* instance types (`m1.medium`, `m1.large`, etc.) is embedded in Deckhouse.

  - **spec.capacity.cpu**
    integer or string

    Required value

    Node vCPU resources.

    Pattern: `^[0-9]+m?$`

    Example:

    ```
    cpu: 1000m
    ```

  - **spec.capacity.memory**
    integer or string

    Required value

    Node memory resources.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    Example:

    ```
    memory: 1500Mi
    ```

- **spec.flavorName**
  string

  Required value

  Flavor of OpenStack servers.

  Get a list of all available flavors: `openstack flavor list`.

  For all non-master nodes it is advisable to use flavor’s with a local disk. If cloud provider supports local disks they are usually faster and cheaper. The disadvantage of using such flavors is the inability to migrate nodes between hypervisors.

  Flavor create example: `openstack flavor create c4m8d50 --ram 8192 --disk 50 --vcpus 4`

  Example:

  ```
  flavorName: m1.medium
  ```

- **spec.imageName**
  string

  Not required value.

  Image to use while provisioning OpenStack servers.

  Use this command to get a list of available images: `openstack image list`.

  The list of OS and their versions supported by Deckhouse can be found in the documentation (take into account

the Deckhouse version used).

By default: either `OpenStackCloudDiscoveryData` value or `instances.imageName` is used.

Example:

```
imageName: ubuntu-18-04-cloud-amd64
```

- **spec.mainNetwork**
  string

  Not required value.

  Path to the network that VirtualMachines' primary NICs will connect to (default gateway).

  By default: the value from `OpenStackCloudDiscoveryData` is used.

  Example:

  ```
  mainNetwork: k8s-msk-178
  ```

- **spec.rootDiskSize**
  integer

  Not required value.

  The size of a root disk (in gigabytes). This parameter also has influence on type of volume that will be used for root disk; the "How to use rootDiskSize and when it is preferred" section describes how to use it.

  Example:

  ```
  rootDiskSize: 20
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

FAQ

> **i** The module lifecycle stage: General Availability

How do I set up LoadBalancer?

> **⚠** To correctly determine the client IP address, use a LoadBalancer with Proxy Protocol support.

It is recommended to limit the list of nodes added to the load balancer pool using the `loadbalancer.openstack.org/node-selector` annotation.

Without a `node-selector` restriction, cloud-controller-manager may use all suitable cluster nodes as load balancer targets. As a result, adding or removing nodes that are not related to the workload served by the load balancer may trigger an update of the load balancer pool membership. In large or frequently changing clusters, such updates may occur regularly and, in some configurations, may cause brief disruptions to existing connections.

It is recommended to use `loadbalancer.openstack.org/node-selector` to select only the nodes that should be used as targets for the corresponding LoadBalancer.

IngressNginxController example

In this example, the Ingress controller pods are scheduled on frontend nodes, while the `loadbalancer.openstack.org/node-selector` annotation limits the load balancer pool to the same nodes:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancerWithProxyProtocol
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/node-selector: "node-role.deckhouse.io/frontend="
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
```

How do I set up security policies on cluster nodes?

There may be many reasons why you may need to restrict or expand incoming/outgoing traffic on cluster VMs in OpenStack:

- Allow VMs on a different subnet to connect to cluster nodes.
- Allow connecting to the ports of the static node so that the application can work.
- Restrict access to external resources or other VMs in the cloud for security reasons.

For all this, additional security groups should be used. You can only use security groups that are created in the cloud tentatively.

Enabling additional security groups on static and master nodes

This parameter can be set either in an existing cluster or when creating one. In both cases, additional security groups are declared in the `OpenStackClusterConfiguration`:

- for master nodes, in the `additionalSecurityGroups` of the `masterNodeGroup` section;
- for static nodes, in the `additionalSecurityGroups` field of the `nodeGroups` subsection that corresponds to the target nodeGroup.

The `additionalSecurityGroups` field contains an array of strings with security group names.

Enabling additional security groups on ephemeral nodes

You have to set the `additionalSecurityGroups` parameter for all OpenStackInstanceClasses in the cluster that require additional security groups. See the parameters of the cloud-provider-openstack module.

How do I create NodeGroups in availability zones?

An OpenStack cluster is deployed in a single region, which is set by the `provider.region` parameter of the OpenStackClusterConfiguration resource. Nodes can be created only in availability zones of this region. Using zones from other regions is not supported.

To get the list of availability zones in the region, run:

```
openstack --os-region-name <REGION> availability zone list --compute
```

CloudPermanent node groups are defined in the `nodeGroups` section of the OpenStackClusterConfiguration resource. To limit availability zones, use the `nodeGroups[].zones` parameter.

CloudEphemeral nodes are created with a separate NodeGroup resource with `nodeType: CloudEphemeral`. Availability zones for them are set in the `spec.cloudInstances.zones` parameter. If the `zones` parameter is specified in OpenStackClusterConfiguration, the required zones must also be added to it.

> **i** If the admission webhook returns the `unknown zone` error when creating a NodeGroup, make sure that the specified zone belongs to the `provider.region` region and, when using the `zones` parameter, is included in the OpenStackClusterConfiguration zones list.

Nodes from another region can be added to the cluster only manually as Static nodes. The `cloud-provider-openstack` module does not create such nodes.

> **⚠** After changing OpenStackClusterConfiguration, run the `dhctl converge` command. For details, see Adding and managing cloud nodes.

Example NodeGroup for CloudEphemeral nodes:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: workers
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    classReference:
      kind: OpenStackInstanceClass
      name: workers
    minPerZone: 1
    maxPerZone: 1
    zones:
    - eu-3a
    - eu-3b
```

How do I create a hybrid cluster?

A hybrid cluster combines bare metal and OpenStack nodes. To create such a cluster, you need an L2 network between all nodes of the cluster.

> **i** The Deckhouse Kubernetes Platform allows to set a prefix for the names of CloudEphemeral nodes added to a hybrid cluster with Static master nodes. To do this, use the `instancePrefix` parameter of the `node-manager` module. The prefix specified in the parameter will be added to the name of all CloudEphemeral nodes added to the cluster. It is not possible to set a prefix for a specific NodeGroup.

To set up a hybrid cluster, follow these steps:

1. Delete flannel from kube-system: `d8 k -n kube-system delete ds flannel-ds`.
2. Enable and configure the module.
3. Create one or more OpenStackInstanceClass custom resources.
4. Create one or more NodeManager custom resources for specifying the number of machines and managing the provisioning process in the cloud.

> **i** **Caution!** Cloud-controller-manager synchronizes OpenStack and Kubernetes states by deleting Kubernetes nodes that are not in OpenStack. In a hybrid cluster, such behavior does not always make sense. That is why cloud-controller-manager automatically skips Kubernetes nodes that do not have the `--cloud-provider=external` parameter (Deckhouse inserts `static://` into nodes in `.spec.providerID`, and cloud-controller-manager ignores them).

Attaching storage devices to instances in a hybrid cluster

To use PersistentVolumes on OpenStack nodes, you must create StorageClass with the appropriate OpenStack volume type. The `openstack volume type list` command lists all available types.

Here is the example config for the `ceph-ssd` volume type:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ceph-ssd
provisioner: csi-cinderplugin # have to be like this
parameters:
  type: ceph-ssd
volumeBindingMode: WaitForFirstConsumer
```

How do I create an image in OpenStack?

1. Download the latest stable Ubuntu 18.04 image:

   ```
   curl -L https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img --output ~/ubuntu-18-04-cloud-amd64
   ```

2. Prepare an OpenStack RC (openrc) file containing credentials for accessing the OpenStack API:

   > **i** The interface for getting an openrc file may differ depending on the OpenStack provider. If the provider has a standard interface for OpenStack, you can download the openrc file using the following instruction.

3. Otherwise, install the OpenStack client using this instruction.

Also, you can run the container and mount an openrc file and a downloaded Ubuntu image in it:

```
docker run -ti --rm -v ~/ubuntu-18-04-cloud-amd64:/ubuntu-18-04-cloud-amd64 -v ~/.openrc:/openrc jmcvea/openstack-client
```

4. Initialize the environment variables from the openrc file:

```
source /openrc
```

5. Get a list of available disk types:

```
/ # openstack volume type list
+--------------------------------------+--------------+-----------+
| ID                                   | Name         | Is Public |
+--------------------------------------+--------------+-----------+
| 8d39c9db-0293-48c0-8d44-015a2f6788ff | ko1-high-iops | True      |
| bf800b7c-9ae0-4cda-b9c5-fae283b3e9fd | dp1-high-iops | True      |
| 74101409-a462-4f03-872a-7de727a178b8 | ko1-ssd       | True      |
| eadd8860-f5a4-45e1-ae27-8c58094257e0 | dp1-ssd       | True      |
| 48372c05-c842-4f6e-89ca-09af3868b2c4 | ssd           | True      |
| a75c3502-4de6-4876-a457-a6c4594c067a | ms1           | True      |
| ebf5922e-42af-4f97-8f23-716340290de2 | dp1           | True      |
| a6e853c1-78ad-4c18-93f9-2bba317a1d13 | ceph          | True      |
+--------------------------------------+--------------+-----------+
```

6. Create an image, pass the disk format to use (if OpenStack does not support local disks or these disks don’t fit):

```
openstack image create --private --disk-format qcow2 --container-format bare \
  --file /ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=dp1-high-iops ubuntu-18-04-cloud-amd64
```

7. Check that the image was created successfully:

```
/ # openstack image show ubuntu-18-04-cloud-amd64
+------------------+------------------------------------
```

are supported.

How to set up online disk resize

The OpenStack API states that the resize is completed successfully. However, Nova does not get any information about the resize from Cinder. As a result, the size of the disk in the guest OS remains the same.

To get rid of this problem, you need to insert the Nova API access parameters into the `cinder.conf` file, e.g., as follows:

```
[nova]
interface = admin
insecure = {{ keystone_service_internaluri_insecure | bool }}
auth_type = {{ cinder_keystone_auth_plugin }}
auth_url = {{ keystone_service_internaluri }}/v3
password = {{ nova_service_password }}
project_domain_id = default
project_name = service
region_name = {{ nova_service_region }}
user_domain_id = default
username = {{ nova_service_user_name }}
```

Source...

How to use `rootDiskSize` and when it is preferred?

Disks in OpenStack

The node disk can be local or network. A local disk in OpenStack, is an ephemeral disk, and a network disk is a persistent disk (cinder storage). Nodes with local disks cannot migrate between hypervisors.

- A network disk is preferred for the master node so that the node can migrate between hypervisors.
- A local disk is preffered for the ephemeral node to save on cost. Not all cloud providers support the use of local disks. If local disks are not supported, you have to use network disks for ephemeral nodes.

Local disk (ephemeral)	Network disk (persistent)
Cheaper	More expensive
Suitable for ephemeral nodes	Suitable for master nodes

The `rootDiskSize` parameter

The `OpenStackInstanceClass` has a `rootDiskSize` parameter, and OpenStack flavor has a disk size parameter. Which disk will be ordered depending on the combination of parameters is shown in the table:

	flavor disk size = 0	flavor disk size > 0
`rootDiskSize` is not specified	*You need to set the size*. Without specifying the size, there will be an error creating a VM.	Local disk with size according to the flavor
`rootDiskSize` is specified	Network disk with the `rootDiskSize` size	Network disk (rootDiskSize) and local disk (according to the flavor). Avoid using this option, as the cloud provider will charge for both disks.

> **i** Please note, that to create a node with the `CloudEphemeral` type in a zone other than zone A, you must first create a flavor with a disk of the required size. The rootDiskSize parameter does not need to be specified.

Network disk is recommended for master nodes and bastion host

- Use flavor with a zero disk size.
- Set the `rootDiskSize` in the `OpenStackInstanceClass`.
- Check the disk type. The disk type will be taken from the OS image if it is set. If it is not set, the disk type will be taken from volumeTypeMap.

Local disk is recommended for ephemeral nodes

- Use flavor with the specified disk size.
- Do not use the `rootDiskSize` parameter in the `OpenStackInstanceClass`.
- Check the disk type. The disk type will be taken from the OS image if it is set. If it is not set, the default disk type of the cloud provider will be used.

How do I check the disk volume in a flavor?

```
# openstack flavor show m1.medium-50g -c disk
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 50    |
+-------+-------+
```

How to override a default volume type of cloud provider?

If there are several types of disks in a *cloud provider*, you can set a default disk type for the image in order to select a specific VM's disk type. To do this, specify the name of a disk type in the image metadata.

Also, you may need to create a custom OpenStack image; the "How do I create an image in OpenStack" section describes how to do it

Example:

```
openstack volume type list
openstack image set ubuntu-18-04-cloud-amd64 --property cinder_img_volume_type=VOLUME_NAME
```

OFFLINE disk resize

Some cloud providers may not support ONLINE disk resizing. If you get the following error, then you need to reduce the number of StatefulSet replicas to 0, wait for disk resizing and return the number of replicas that was before the start of the operation.

```
Warning  VolumeResizeFailed     5s (x11 over 41s)  external-resizer cinder.csi.openstack.org
resize volume "pvc-555555-ab66-4f8d-947c-296520bae4c1" by resizer "cinder.csi.openstack.org" failed:
rpc error: code = Internal desc = Could not resize volume "bb5a275b-3f30-4916-9480-9efe4b6dfba5" to size 2:
Expected HTTP response code [202] when accessing
[POST https://public.infra.myfavourite-cloud-provider.ru:8776/v3/555555555555/volumes/bb5a275b-3f30-4916-9480-
9efe4b6dfba5/action], but got 406 instead
{"computeFault": {"message": "Version 3.42 is not supported by the API. Minimum is 3.0 and maximum is 3.27.", "code": 406}}
```

What to do if switching to nodes in lower-priority groups takes a long time?

If switching to nodes in lower-priority groups takes a long time, follow the instructions.

Examples

> **i** The module lifecycle stage: General Availability

Below are two simple examples of OpenStack cloud provider configuration.

Example 1

```
apiVersion: deckhouse.io/v1
kind: OpenStackInstanceClass
metadata:
  name: test
spec:
  flavorName: m1.large
```

Example 2

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-openstack
spec:
  version: 1
  enabled: true
  settings:
    connection:
      authURL: https://test.tests.com:5000/v3/
      domainName: default
      tenantName: default
      username: jamie
      password: nein
      region: SomeRegion
    externalNetworkNames:
    - public
    internalNetworkNames:
    - kube
    instances:
      sshKeyPairName: my-ssh-keypair
      securityGroups:
      - default
      - allow-ssh-and-icmp
    zones:
    - zone-a
    - zone-b
    tags:
      project: cms
      owner: default
```

Cloud provider - OpenStack: Layouts

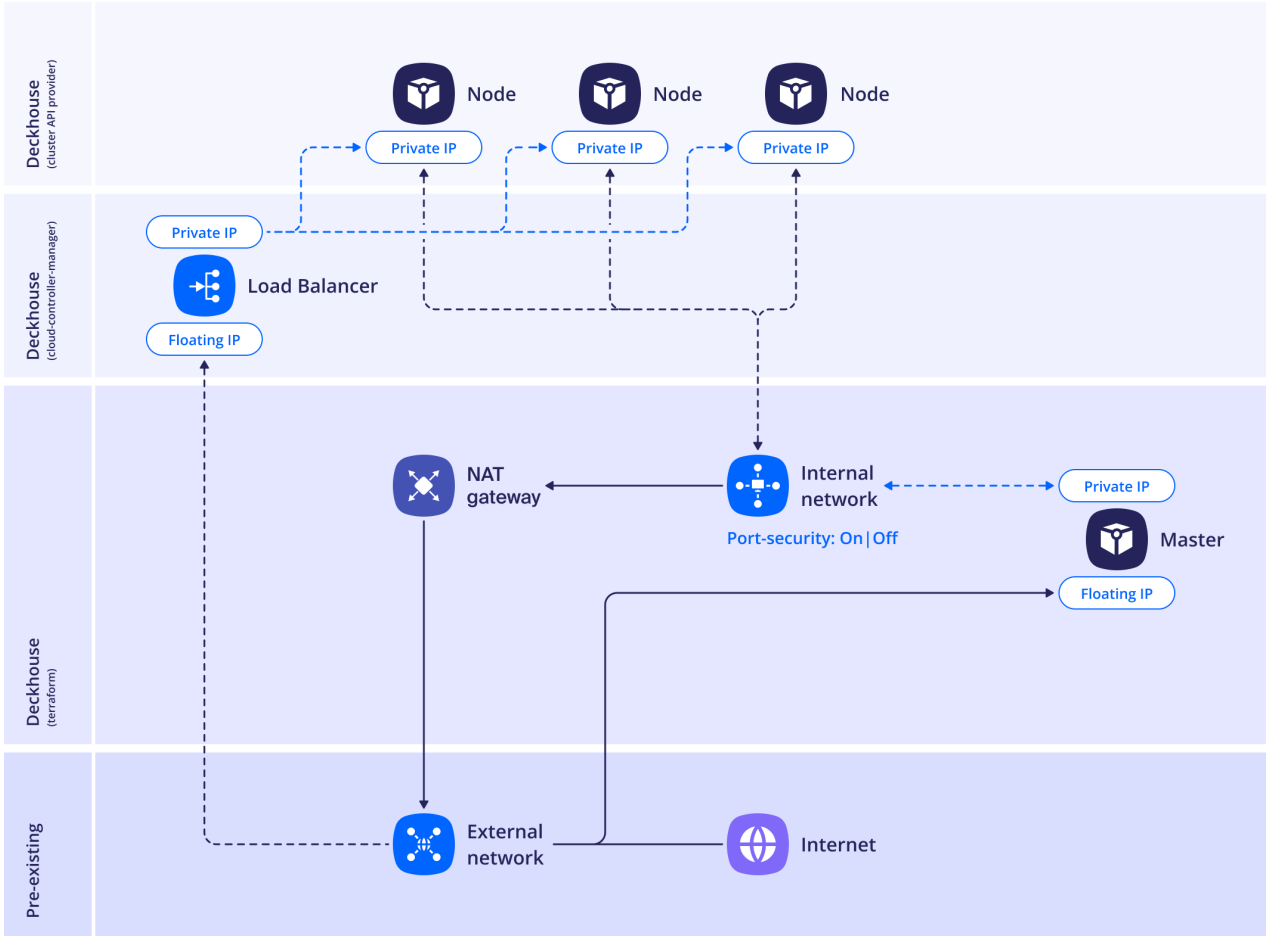
> **i** The module lifecycle stage: General Availability

Four layouts are supported. Below is more information about each of them.

Standard

In this scheme, an internal cluster network is created with a gateway to the public network; the nodes do not have public IP addresses. Note that the floating IP is assigned to the master node.

> **i** **Caution!** If the provider does not support SecurityGroups, all applications running on nodes with Floating IPs assigned will be available at a public IP. For example, `kube-apiserver` on master nodes will be available on port 6443. To avoid this, we recommend using the SimpleWithInternalNetwork placement strategy or Standard strategy with bastion host.



Example of the layout configuration:

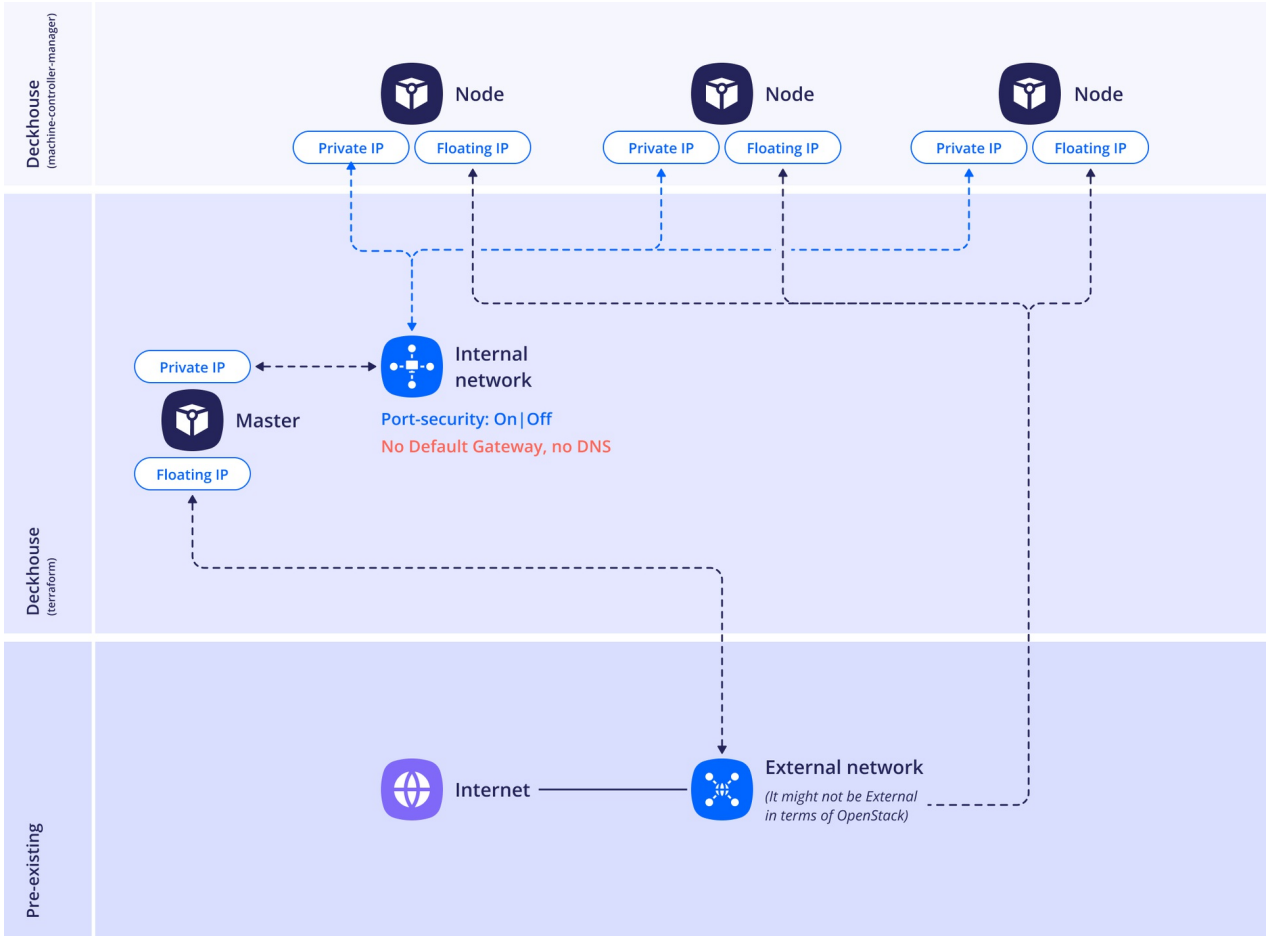
```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Standard
standard:
  internalNetworkCIDR: 192.168.199.0/24       # Required.
  internalNetworkDNSServers:                 # Required.
  - 8.8.8.8
  - 4.2.2.2
  internalNetworkSecurity: true|false         # Optional, default true.
  externalNetworkName: shared                # Required.
  bastion:
    zone: ru2-b                             # Optional.
    volumeType: fast-ru-2b                   # Optional.
    instanceClass:
      flavorName: m1.large                   # Required.
      imageName: ubuntu-20-04-cloud-amd64      # Required.
      rootDiskSize: 50                      # Optional, default 50 gigabytes.
      additionalTags:
        severity: critical                   # Optional.
        environment: production              # Optional.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
    additionalTags:
      severity: critical
      environment: production
  # Required, volume type map for etcd and kubernetes certs
  # (always use fastest disk supplied by provider).
  volumeTypeMap:
```

```
    # If rootDiskSize specified than this volume type will
    # be also used for master root volume.
    ru-1a: fast-ru-1a
    ru-1b: fast-ru-1b
    ru-1c: fast-ru-1c
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                  # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, default false, determines if config drive is
    # required during vm bootstrap process. It's needed if there
    # is no dhcp in network that is used as default gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                   # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  zones:
  - ru-1a
  - ru-1b
sshPublicKey: "<SSH_PUBLIC_KEY>"
tags:
  project: cms
  owner: default
provider:
  ...
```

StandardWithNoRouter

An internal cluster network is created that does not have access to the public network. All nodes (including master ones) have two interfaces: the first one to the public network, the second one to the internal network. This layout should be used if you want all nodes in the cluster to be directly accessible.

> **i**
>
> **Caution!** This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a load balancer with the Floating IP. An internal load balancer with the virtual IP in the public network is only accessible to cluster nodes.
>
> **Caution!** In this strategy, it is necessary to explicitly specify the name of the internal network in `additionalNetworks` when creating an `OpenStackInstanceClass` in the cluster.



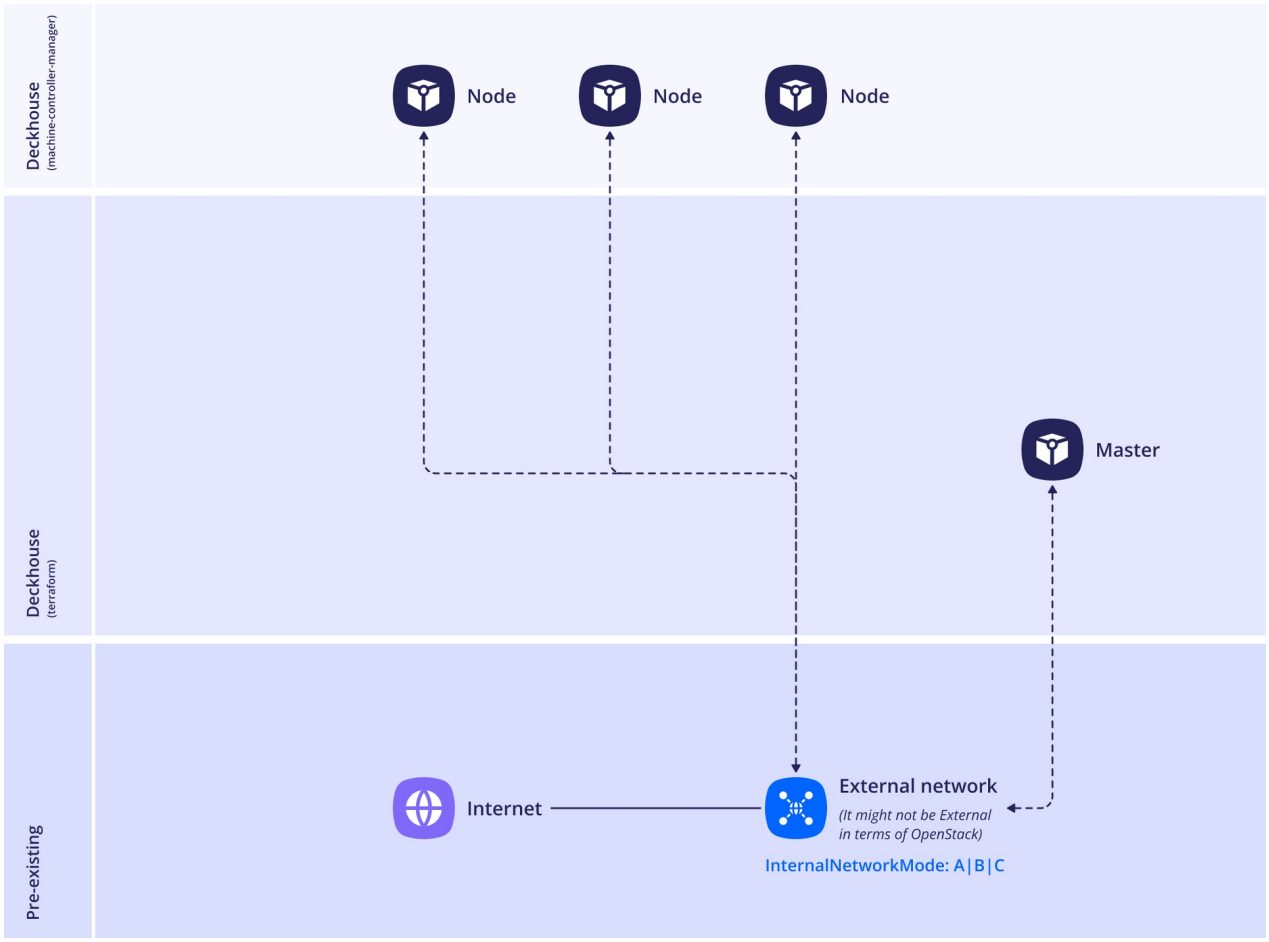
Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: StandardWithNoRouter
standardWithNoRouter:
  internalNetworkCIDR: 192.168.199.0/24        # Required.
  externalNetworkName: ext-net                # Required.
  # Optional, whether dhcp is enabled in specified external
  # network (default true).
  externalNetworkDHCP: false
  internalNetworkSecurity: true|false          # Optional, default true.
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and kubernetes certs
  # (always use fastest disk supplied by provider).
  volumeTypeMap:
    # If rootDiskSize specified than this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                     # Required.
    imageName: ubuntu-18-04-cloud-amd64        # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, default false, determines if config drive
    # is required during vm bootstrap process. It's needed
    # if there is no dhcp in network that is used as default
    # gateway.
    configDrive: false
    # Required, the gateway of the network will be used as the default gateway.
    # Matches the cloud.prefix in the ClusterConfiguration resource.
    mainNetwork: kube
    additionalNetworks:                      # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required if rootDiskSize is specified. Volume type map for node's root volume
  volumeTypeMap:
    nova: ceph-ssd
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Simple

The master node and cluster nodes are connected to the existing network. This placement strategy might come in handy if you need to merge a Kubernetes cluster with existing VMs.

> **Caution!** This strategy does not support a LoadBalancer since a Floating IP is not available for the router-less network. Thus, you cannot provision a load balancer with the Floating IP. An internal load balancer with the virtual IP in the public network is only accessible to cluster nodes.



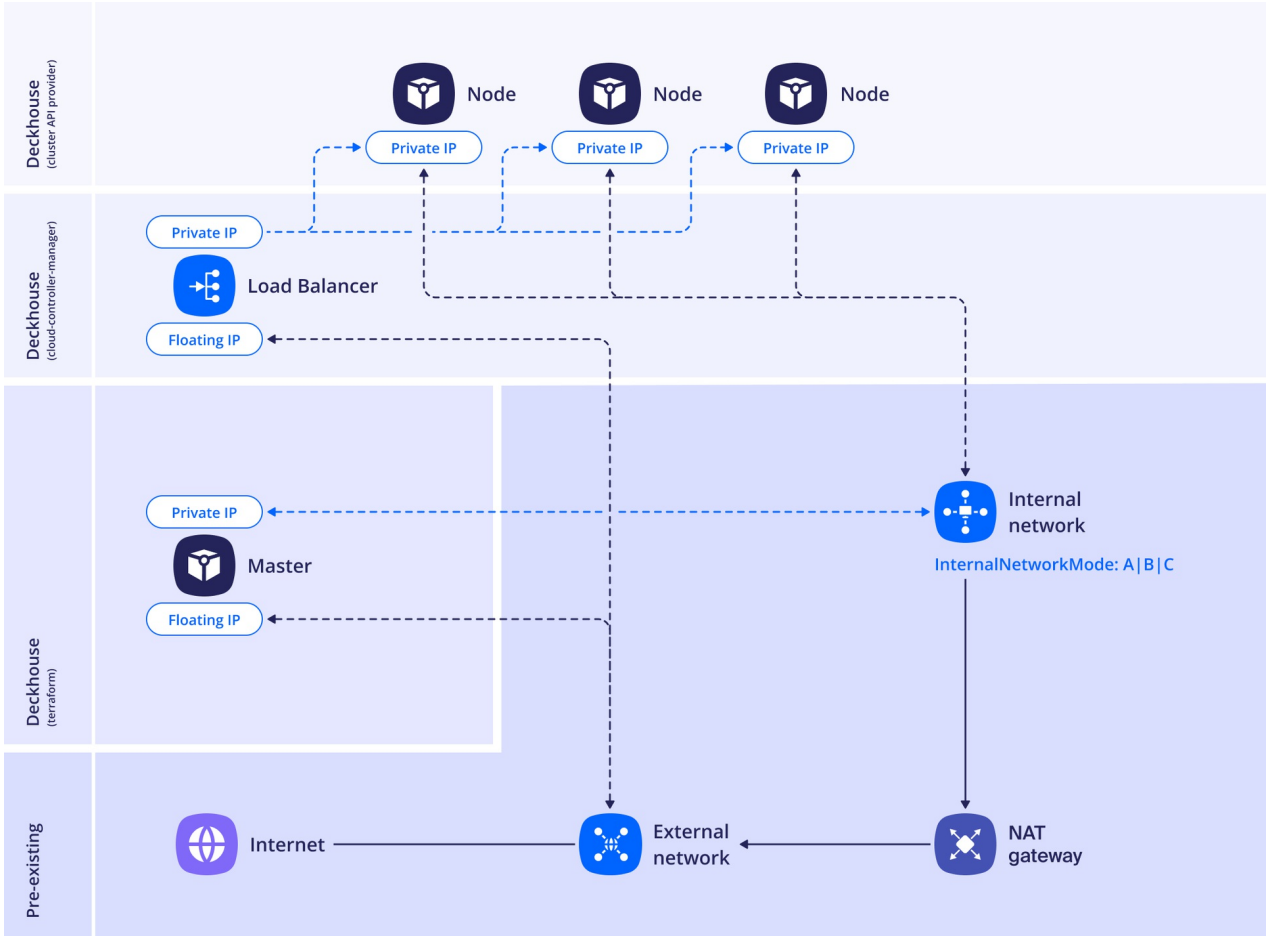
Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: Simple
simple:
  externalNetworkName: ext-net              # Required.
  externalNetworkDHCP: false                # Optional, default true.
  # Optional, by default VXLAN, may also be DirectRouting
  # or DirectRoutingWithPortSecurityEnabled.
  podNetworkMode: VXLAN
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and kubernetes certs
  # (always use fastest disk supplied by provider).
  volumeTypeMap:
    # If rootDiskSize specified than this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, default false, determines if config drive
    # is required during vm bootstrap process. It's needed
    # if there is no dhcp in network that is used as default
    # gateway.
    configDrive: false
    # Required, network will be used as default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

SimpleWithInternalNetwork

The master node and cluster nodes are connected to the existing network. This placement strategy might come in handy if you need to merge a Kubernetes cluster with existing VMs.

> **i** **Caution!** This placement strategy does not involve the management of `SecurityGroups` (it is assumed they were created beforehand). To configure security policies, you must explicitly specify both `additionalSecurityGroups` in the `OpenStackClusterConfiguration` for the masterNodeGroup and other nodeGroups, and `additionalSecurityGroups` when creating `OpenStackInstanceClass` in the cluster.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: OpenStackClusterConfiguration
layout: SimpleWithInternalNetwork
simpleWithInternalNetwork:
  # Required, all cluster nodes have to be in the same subnet.
  internalSubnetName: pivot-standard
  # Optional, by default DirectRoutingWithPortSecurityEnabled,
  # may also be DirectRouting or VXLAN.
  podNetworkMode: DirectRoutingWithPortSecurityEnabled
  # Optional, if set will be used for load balancer default
  # configuration and ordering master floating IP.
  externalNetworkName: ext-net
  # Optional, default value is true.
  masterWithExternalFloatingIP: false
masterNodeGroup:
  replicas: 3
  instanceClass:
    flavorName: m1.large                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 50
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
  # Required, volume type map for etcd and kubernetes certs
  # (always use fastest disk supplied by provider).
  volumeTypeMap:
    # If rootDiskSize specified than this volume type will
    # be also used for master root volume.
    nova: ceph-ssd
nodeGroups:
- name: front
  replicas: 2
  instanceClass:
    flavorName: m1.small                    # Required.
    imageName: ubuntu-18-04-cloud-amd64       # Required.
    # Optional, local disk is used if not specified.
    rootDiskSize: 20
    # Optional, default false, determines if config drive
    # is required during vm bootstrap process. It's needed
    # if there is no dhcp in network that is used as default
    # gateway.
    configDrive: false
    # Required, network will be used as default gateway.
    # Matches the name of the pre-created network.
    mainNetwork: kube
    additionalNetworks:                     # Optional.
    - office
    - shared
    # Optional, if there are networks with disabled port
    # security their names must be specified.
    networksWithSecurityDisabled:
    - office
    # Optional, list of network pools where to order floating IPs.
    floatingIPPools:
    - public
    - shared
    # Optional, additional security groups.
    additionalSecurityGroups:
    - sec_group_1
    - sec_group_2
sshPublicKey: "<SSH_PUBLIC_KEY>"
provider:
  ...
```

Cloud provider — VMware Cloud Director

> **i** The module lifecycle stage: General Availability

The `cloud-provider-vcd` module integrates Deckhouse Kubernetes Platform with VMware Cloud Director. It allows the

`node-manager` module to use VCD resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-vcd` module:

- Managing VCD resources via `cloud-controller-manager`:
  - updates virtual machine and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in VCD;
  - creates load balancers for Services of the LoadBalancer type. This uses VMware NSX Advanced Load Balancer (Avi); support is available with NSX-T.
- Provisioning disks via the Named Disk CSI driver (`named-disk.csi.cloud-director.vmware.com`) so that PersistentVolumes can be requested from the cluster.
- Provisioning CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-vcd`. Base infrastructure is provisioned depending on the layout.
- Provisioning CloudEphemeral nodes via Cluster API (CAPI). Virtual machine parameters are set in the VCDInstanceClass resource.
- Registering with `node-manager` so that VCDInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

> **i** For VCD API versions earlier than 37.2, the module uses compatibility mode (legacy components).

Configuration

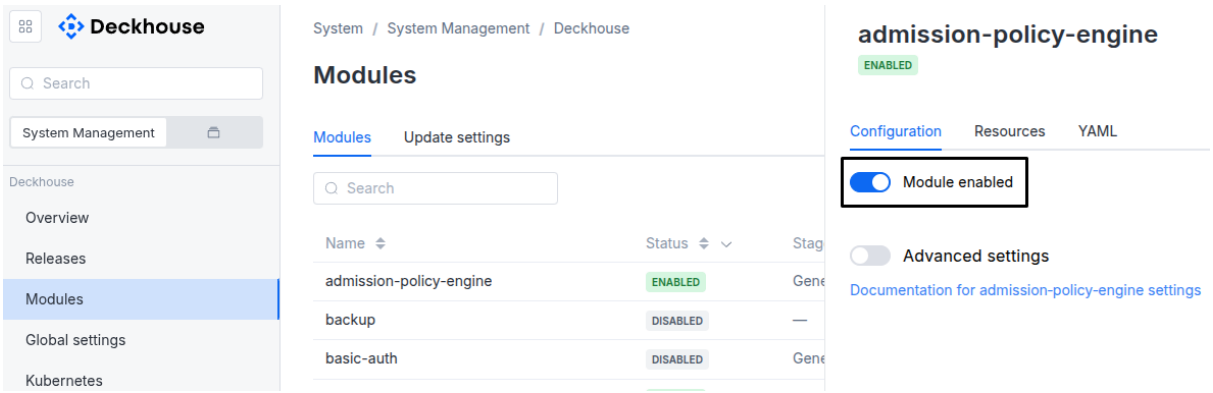
> **i** The module lifecycle stage: General Availability

The module is automatically enabled for all cloud clusters deployed in VMware Cloud Director.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-vcd` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cloud-provider-vcd
```

- **Using ModuleConfig `cloud-provider-vcd`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-vcd` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-vcd`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vcd
spec:
  enabled: true
```

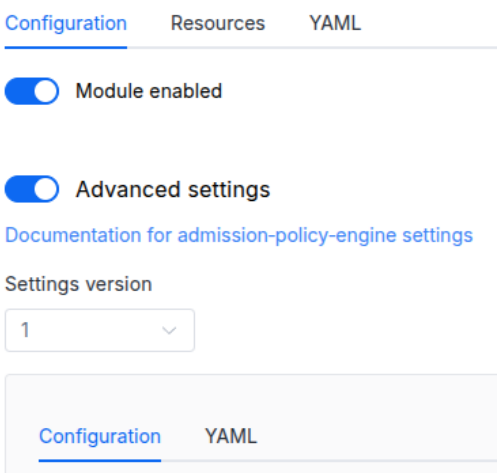
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-vcd` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

Example:



You can also edit the ModuleConfig object `cloud-provider-vcd` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-vcd`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-vcd` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-vcd
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cloud-provider-vcd` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cloud-provider-vcd`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-vcd
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.storageClass**
    object

    The module automatically creates StorageClasses that are available in VCD (VMware Cloud Director).

    - **settings.storageClass.default**
      Deprecated
      string

      The name of StorageClass that will be used in the cluster by default.

      If the parameter is omitted, the default StorageClass is either:

      - an arbitrary StorageClass present in the cluster that has the default annotation;
      - StorageClass for `default StorageProfile`;
      - the first (in lexicographic order) StorageClass created by the module.

      > **i** **Parameter is deprecated.** Instead, use the global parameter global.defaultClusterStorageClass.

Example:

```
default: ceph-ssd
```

- **settings.storageClass.exclude**
  array of strings

  A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

  Example:

```
exclude:
- ".*-hdd"
- iscsi-fast
```

Provider configuration

> **i** The module lifecycle stage: General Availability

> **i** If the cluster control plane is hosted on a virtual machines or bare-metal servers, the cloud provider uses the settings from the `cloud-provider-vcd` module in the Deckhouse configuration. Otherwise, if the cluster control plane is hosted in a cloud, the cloud provider uses the VCDClusterConfiguration structure for configuration.

VCDClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in VCD.

Used by the cloud provider if a cluster's control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

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

> **i** After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: VCDClusterConfiguration
sshPublicKey: "<SSH_PUBLIC_KEY>"
organization: My_Org
virtualDataCenter: My_Org
virtualApplicationName: Cloud
mainNetwork: internal
layout: Standard
internalNetworkCIDR: 172.16.2.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    template: Templates/ubuntu-focal-20.04
    sizingPolicy: 4cpu8ram
    rootDiskSizeGb: 20
    etcdDiskSizeGb: 20
    storageProfile: nvme
nodeGroups:
- name: worker
  replicas: 1
  instanceClass:
    template: Org/Templates/ubuntu-focal-20.04
    sizingPolicy: 16cpu32ram
    storageProfile: ssd
provider:
  server: "<SERVER>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  insecure: true
```

- **apiVersion**

string

Required value

Allowed values: `deckhouse.io/v1`

- **bastion**
  object

  Parameters for the bastion host, which is used to access the cluster’s nodes.

  - **bastion.instanceClass**
    object

    Required value

    Partial contents of the fields of the VCDInstanceClass.

    - **bastion.instanceClass.additionalMetadata**
      object

      Custom string key-value pairs to be stored in the Metadata of the bastion, useful for organizing and categorizing your virtualization inventory.

      The properties in this field will be merged with the existing metadata stored in the `metadata` field at the root of the spec. This field will take precedence over the existing metadata in cases of duplicate keys.

    - **bastion.instanceClass.mainNetworkIPAddress**
      string

      The IP address of the bastion host in the `mainNetwork`. If not set, the DHCP client will be enabled.

      Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

    - **bastion.instanceClass.placementPolicy**
      string

      Placement policy to be used on this machine.

      If no placement policy is specified, the default placement policy will be used to create the bastion host.

      Length: `1..128`

    - **bastion.instanceClass.rootDiskSizeGb**
      integer

      Root disk size in gigabytes.

      Default: `50`

      Allowed values: `0 <= X`

      Example:

      ```
      rootDiskSizeGb: 50
      ```

    - **bastion.instanceClass.sizingPolicy**
      string

      Required value

      Sizing policy to be used on this machine.

      If no sizing policy is specified, the default sizing policy will be used to create the bastion host.

      Length: `1..128`

    - **bastion.instanceClass.storageProfile**
      string

      Required value

      Storage profile to be used on this machine.

      Length: `1..128`

    - **bastion.instanceClass.template**

      Required value

      Path of the template OVA that is to be used.

      Consists of organization (optional), catalog, template name or template identifier. If a shared catalog is used, the organization name must be specified (it may differ from the value of `provider.organization`).

      If you have templates with the same name in multiple catalogs with the same name, you should specify the identifier of the template.

      You can find the template identifier in the URL when you open the template in the VCD UI. For example, if the URL is `https://vcd.example.com/tenant/MyOrg/libraries/catalogs/1032506a-4555-43e1-8589-77b0c0522c75/catalog-vapp-templates/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7/general`, then the template identifier is `5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7`.

Length: `1..128`

- **createDefaultFirewallRules**
  boolean

  If enabled, configures the following default firewall rules for the cluster’s main network (mainNetwork):

  - Allows all outgoing connections from mainNetwork.
  - Allows all incoming ICMP packets to mainNetwork.
  - Allows all incoming TCP connections to mainNetwork on port 22 (SSH).
  - Allows all incoming TCP and UDP connections to mainNetwork for NodePorts in the range 30000–32767.

  Attention! Using this option together with an Edge Gateway of type NSX-T implies that only one cluster should be deployed per Edge Gateway, because in this case all existing firewall rules will be overwritten.

  Default: `false`

- **edgeGateway**
  object

  VMware Cloud Director Edge Gateway parameters (belongs to Virtual Data Center).

  - **edgeGateway.NSX-V**
    object

    Additional parameters required if `type` is NSX-V.

    - **edgeGateway.NSX-V.externalNetworkName**
      string

      External network name for build DNAT rule for master node. The external network is the one connected to the Edge Gateway in `Gateway Interface` and has an external IP address.

      Length: `1..128`

    - **edgeGateway.NSX-V.externalNetworkType**
      string

      External network type for build DNAT rule for master node (one of “org” or “ext”).

      Allowed values: `org`, `ext`

  - **edgeGateway.externalIP**
    string

    Required value

    The IP address accessible from external networks through which incoming SSH connections are forwarded to the master node using DNAT.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  - **edgeGateway.externalPort**
    integer

    The external port mapped by the DNAT rule to forward incoming SSH connections to the master node.

    Default: `22`

    Allowed values: `1 <= X <= 65535`

    Example:

    ```
    externalPort: 22
    ```

  - **edgeGateway.name**
    string

    Required value

    VMware Cloud Director Edge Gateway name.

    Length: `1..128`

  - **edgeGateway.type**
    string

    Required value

    Network virtualization platform type providing the VMware Cloud Director Edge Gateway (one of “NSX-V” or “NSX-T”). The type can be obtained from your administrator.

    Allowed values: `NSX-V`, `NSX-T`

- **internalNetworkCIDR**
  string

  Subnet for master nodes in the internal network.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **internalNetworkDHCPPoolStartAddress**
  integer

  Specifies the starting address number in the internal network from which the DHCP address pool will begin. Addresses below this number will not be assigned automatically by DHCP and are available for manual allocation. If not set, the default start address is 30.

  For example, in a network 192.168.100.128/27, a value of 10 means the DHCP pool will start with the IP address 192.168.100.138. The pool will end at the address immediately before the broadcast address (192.168.100.159), i.e., the last assignable host address.

  Allowed values: `2 <= X`

- **internalNetworkDNSServers**
  array of strings

  A list of addresses of the recursive DNSs servers.

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **kind**
  string

  Required value

  Allowed values: `VCDClusterConfiguration`

- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

- **legacyMode**
  boolean

  Support for outdated versions of the VCD API.

  If set to `true`, the API version below `37.2` will be used.

  The parameter has an effect only when installing a cluster. In the working cluster, the need to enable the support for outdated versions of the VCD API is determined automatically.

  Default: `false`

- **mainNetwork**
  string

  Required value

  Path to the network that VirtualMachines' primary NICs will connect to (default gateway).

  Example:

  ```
  mainNetwork: internal
  ```

- **masterNodeGroup**
  object

  Required value

  The definition of the master's NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the fields of the VCDInstanceClass.

    - **masterNodeGroup.instanceClass.additionalMetadata**
      object

      Custom string key-value pairs to be stored in the Metadata of every master node VM, useful for organizing and categorizing your virtualization inventory.

      The properties in this field will be merged with the existing metadata stored in the `metadata` field at the root of the spec. This field will take precedence over the existing metadata in cases of duplicate keys.

    - **masterNodeGroup.instanceClass.affinityRule**

object

Specifies the affinity rule applied to virtual machines of the master nodes within the cluster. Affinity rules determine how nodes are scheduled in relation to each other at the hypervisor level, which can help improve fault tolerance, and control the distribution of workloads. If not specified, no affinity rule will be applied by default.

- **masterNodeGroup.instanceClass.affinityRule.polarity**
  string

  Required value

  The polarity of the affinity rule. Must be either `Affinity` or `AntiAffinity`. `Affinity` means that nodes should be placed on the same host, while `AntiAffinity` means they should be placed on different hosts.

  Allowed values: `Affinity`, `AntiAffinity`

- **masterNodeGroup.instanceClass.affinityRule.required**
  boolean

  Indicates whether the affinity rule is mandatory (`true`) or only preferred (`false`). If set to `true`, the rule must be enforced; if `false`, the rule is applied if possible but not strictly required.

  Default: `false`

- **masterNodeGroup.instanceClass.etcdDiskSizeGb**
  integer

  Etcd disk size in gigabytes.

  Default: `10`

  Example:

  ```
  etcdDiskSizeGb: 10
  ```

- **masterNodeGroup.instanceClass.mainNetworkIPAddresses**
  array of strings

  A list of static IP addresses (with a CIDR prefix) sequentially allocated to nodes in the `mainNetwork`.

  By default, the DHCP client is enabled.

  Example:

  ```
  mainNetworkIPAddresses:
  - 10.1.14.20
  - 10.1.4.21
  ```

  - **Element of the array**
    string

    An IP address with a CIDR prefix.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **masterNodeGroup.instanceClass.placementPolicy**
  string

  PlacementPolicy is the placement policy to be used on this machine.

  If no placement policy is specified, default placement policy will be used to create the nodes.

  Example:

  ```
  placementPolicy: Hypervisor1
  ```

- **masterNodeGroup.instanceClass.rootDiskSizeGb**
  integer

  Root disk size in gigabytes.

  Default: `50`

  Example:

  ```
  rootDiskSizeGb: 50
  ```

- **masterNodeGroup.instanceClass.sizingPolicy**
  string

  Required value

  SizingPolicy is the sizing policy to be used on this machine.

  If no sizing policy is specified, default sizing policy will be used to create the nodes.

  Example:

```
sizingPolicy: 4Cpu8Memory
```

- **masterNodeGroup.instanceClass.storageProfile**
  string

  Required value

  StorageProfile is the storage profile to be used on this machine.

  Example:

  ```
  storageProfile: ssd
  ```

- **masterNodeGroup.instanceClass.template**
  string

  Required value

  Path of the template OVA that is to be used.

  Consists of organization (optional), catalog, template name or template identifier. If a shared catalog is used, the organization name must be specified (it may differ from the value of `provider.organization`).

  If you have templates with the same name in multiple catalogs with the same name, you should specify the identifier of the template.

  You can find the template identifier in the URL when you open the template in the VCD UI. For example, if the URL is `https://vcd.example.com/tenant/MyOrg/libraries/catalogs/1032506a-4555-43e1-8589-77b0c0522c75/catalog-vapp-templates/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7/general`, then the template identifier is `5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7`.

  Examples:

  ```
  template: Linux/ubuntu2204-cloud-ova
  ```

  ```
  template: MyOrg/Linux/ubuntu2204-cloud-ova
  ```

  ```
  template: Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
  ```

  ```
  template: MyOrg/Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
  ```

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **metadata**
  object

  Custom metadata entries to be stored in cluster objects such as virtual machines, disks and network.

  Clearing the `metadata` and `additionalMetadata` fields, if they contain any keys, will result in all metadata no longer being tracked except for nodes of the CloudEphemeral type. This means that if you need to remove metadata entirely, you will have to do it manually.

  > **Warning.** Changing the metadata will cause the recreation of CloudEphemeral nodes.

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the VCDInstanceClass.

    - **nodeGroups.instanceClass.additionalMetadata**
      object

      Custom string key-value pairs to be stored in the Metadata of every node VM, useful for organizing and categorizing your virtualization inventory.

      The properties in this field will be merged with the existing metadata stored in the `metadata` field at the root of the spec. This field will take precedence over the existing metadata in cases of duplicate keys.

    - **nodeGroups.instanceClass.affinityRule**
      object

Specifies the affinity rule applied to virtual machines of nodes of the NodeGroup within the cluster. Affinity rules determine how nodes are scheduled in relation to each other at the hypervisor level, which can help improve fault tolerance, and control the distribution of workloads. If not specified, no affinity rule will be applied by default.

- **nodeGroups.instanceClass.affinityRule.polarity**
  string

  Required value

  The polarity of the affinity rule. Must be either `Affinity` or `AntiAffinity`. `Affinity` means that nodes should be placed on the same host, while `AntiAffinity` means they should be placed on different hosts.

  Allowed values: `Affinity`, `AntiAffinity`

- **nodeGroups.instanceClass.affinityRule.required**
  boolean

  Indicates whether the affinity rule is mandatory (`true`) or only preferred (`false`). If set to `true`, the rule must be enforced; if `false`, the rule is applied if possible but not strictly required.

  Default: `false`

- **nodeGroups.instanceClass.mainNetworkIPAddresses**
  array of strings

  A list of static IP addresses (with a CIDR prefix) sequentially allocated to nodes in the `mainNetwork`.

  By default, the DHCP client is enabled.

  Example:

  ```
  mainNetworkIPAddresses:
  - 10.1.14.20
  - 10.1.4.21
  ```

  - **Element of the array**
    string

    An IP address with a CIDR prefix.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **nodeGroups.instanceClass.placementPolicy**
  string

  PlacementPolicy is the placement policy to be used on this machine.

  If no placement policy is specified, default placement policy will be used to create the nodes.

  Example:

  ```
  placementPolicy: Hypervisor1
  ```

- **nodeGroups.instanceClass.rootDiskSizeGb**
  integer

  Root disk size in gigabytes.

  Default: `50`

  Example:

  ```
  rootDiskSizeGb: 50
  ```

- **nodeGroups.instanceClass.sizingPolicy**
  string

  Required value

  SizingPolicy is the sizing policy to be used on this machine.

  If no sizing policy is specified, default sizing policy will be used to create the nodes.

  Example:

  ```
  sizingPolicy: 4Cpu8Memory
  ```

- **nodeGroups.instanceClass.storageProfile**
  string

  Required value

  StorageProfile is the storage profile to be used on this machine.

  Example:

```
storageProfile: ssd
```

- **nodeGroups.instanceClass.template**
  string

  Required value

  Path of the template OVA that is to be used.

  Consists of organization (optional), catalog, template name or template identifier. If a shared catalog is used, the organization name must be specified (it may differ from the value of `provider.organization`).

  If you have templates with the same name in multiple catalogs with the same name, you should specify the identifier of the template.

  You can find the template identifier in the URL when you open the template in the VCD UI. For example, if the URL is `https://vcd.example.com/tenant/MyOrg/libraries/catalogs/1032506a-4555-43e1-8589-77b0c0522c75/catalog-vapp-templates/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7/general`, then the template identifier is `5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7`.

  Examples:

  ```
  template: Linux/ubuntu2204-cloud-ova
  ```

  ```
  template: MyOrg/Linux/ubuntu2204-cloud-ova
  ```

  ```
  template: Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
  ```

  ```
  template: MyOrg/Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
  ```

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: 'true'
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to cluster resources.

    The same as the `metadata.labels` standard field.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object.

    > **i** Available fields: `effect`, `key`, and `values`.

    Example:

    ```
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
    ```

    - **nodeGroups.nodeTemplate.taints.effect**
      string

Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **nodeGroups.nodeTemplate.taints.key**
  string
- **nodeGroups.nodeTemplate.taints.value**
  string

- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes to create.

- **organization**
  string

  Required value

  VMware Cloud Director Organization name.

- **provider**
  object

  Required value

  Parameters for connecting to the VCD.

  - **provider.apiToken**
    string

    The token for authentication.

    > **Caution!** When using `apiToken`, leave `username` and `password` empty.

  - **provider.insecure**
    boolean

    Set to `true` if VCD has a self-signed certificate.

    Default: `false`

  - **provider.password**
    string

    The user’s password.

  - **provider.server**
    string

    Required value

    The host or the IP address of the VCD server.

  - **provider.username**
    string

    The login ID.

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **virtualApplicationName**
  string

  Required value

  VMware Cloud Director Virtual Application name (belongs to Virtual Data Center).

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **virtualDataCenter**
  string

  Required value

  VMware Cloud Director Virtual Data Center name (belongs to Organization).

Preparing environment

> The module lifecycle stage: General Availability

List of required VCD resources

- Organization
- VirtualDataCenter
- vApp
- StoragePolicy
- SizingPolicy
- Network
- EdgeRouter
- Catalog

The Organization, VirtualDataCenter, StoragePolicy, SizingPolicy, EdgeRouter, and Catalog resources must be provided by your VMware Cloud Director service provider.

⚠ Each VDC (Virtual Data Center) must have an Edge Gateway configured, and the cluster network must be connected to it.

Network (internal network) can be configured either by your VMware Cloud Director service provider or manually by you. If you choose the `WithNAT` placement scheme, the network will be created automatically. The following section describes how to configure the internal network manually.

User permissions

For access to the VMware Cloud Director API, one of the following options is sufficient:

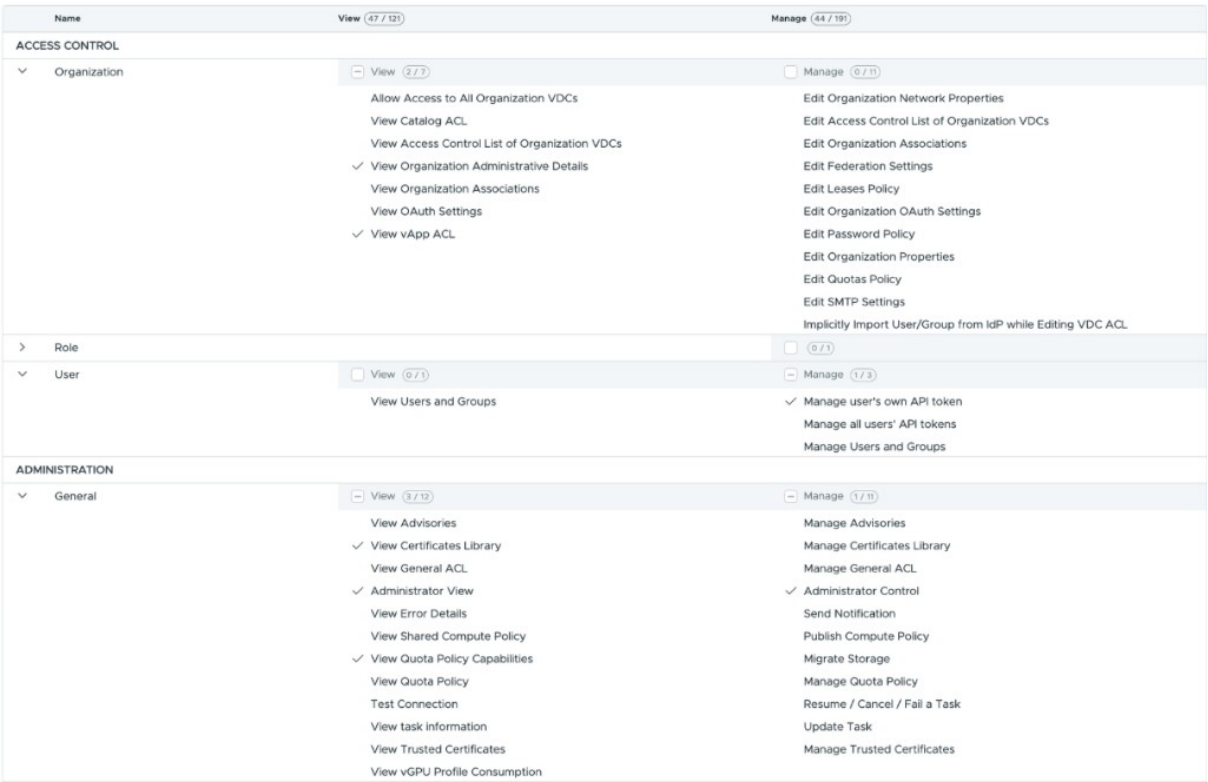
- The `Organization Administrator` role with the `Preserve All ExtraConfig Elements During OVF Import and Export` right. This option does not require configuring a separate role, but grants the user full administrative access to the organization.
- A custom role with the set of rights listed below. This set is sufficient to create a cluster, provision nodes (including CloudEphemeral with static addressing and DHCP), and work with CSI disks and StorageClass without granting additional administrative rights.

In VMware Cloud Director, user permissions come from the role and from a **Rights Bundle** — a set of rights published to the organization. Add the `Preserve All ExtraConfig Elements During OVF Import and Export` right both to the role and to the organization's Rights Bundle.

⚠ Without the `Preserve All ExtraConfig Elements During OVF Import and Export` right in both the role and the organization's Rights Bundle, `userdata` is not passed to the virtual machines of ephemeral nodes.

To create a custom role, enable the rights as shown below. Right names match the VCD UI.

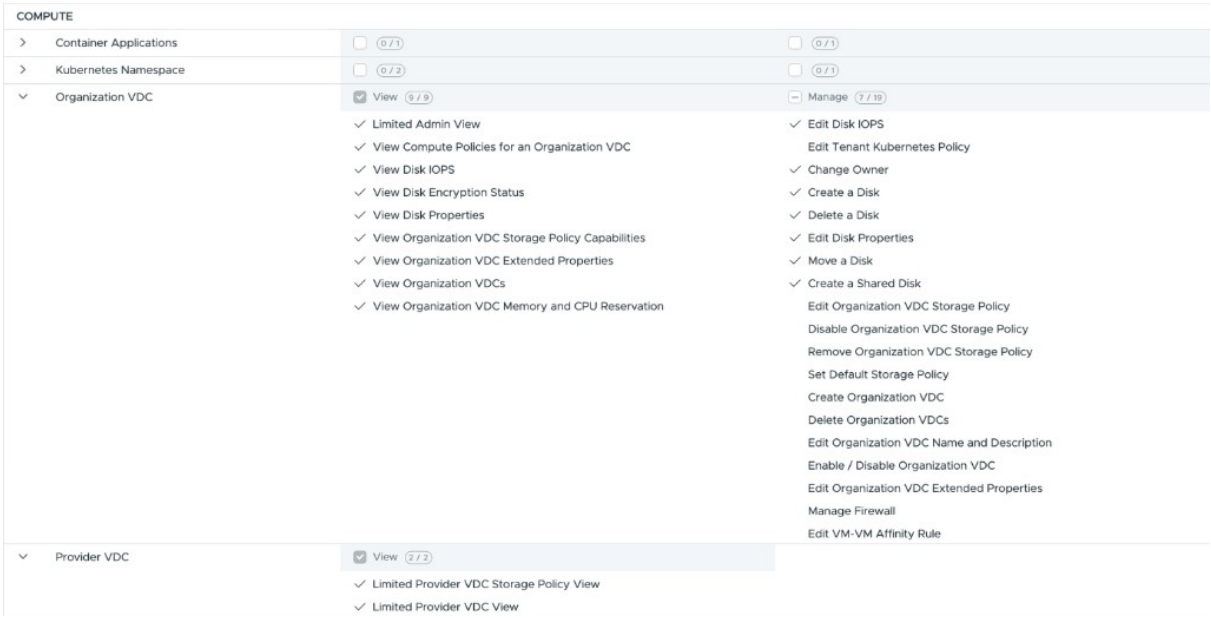
1. In the "ACCESS CONTROL" and "ADMINISTRATION" sections, enable the following rights:
   - Organization → View: `View Organization Administrative Details`, `View vApp ACL`
   - User → Manage: `Manage user's own API token`
   - General → View: `View Certificates Library`, `Administrator View`, `View Quota Policy Capabilities`
   - General → Manage: `Administrator Control`



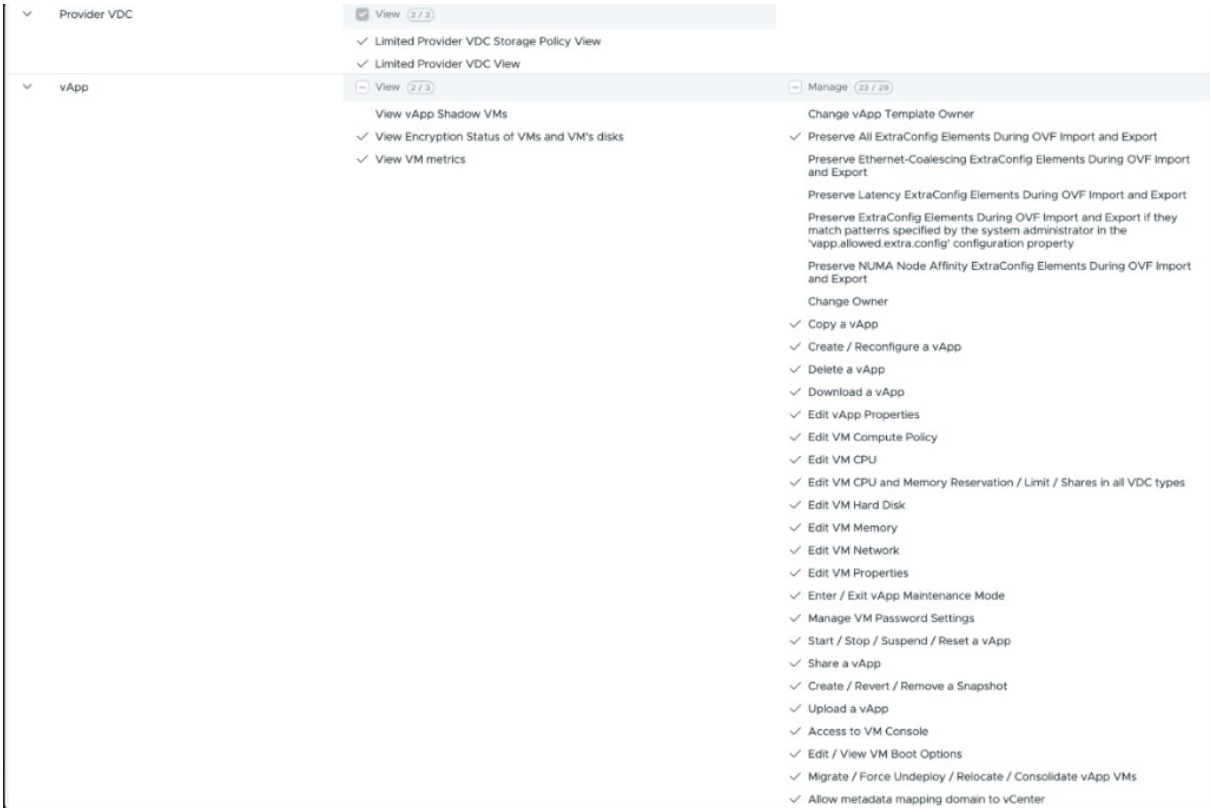
2. In the "COMPUTE" → "Organization VDC" section, enable all View rights and the following Manage rights:
   - `Edit Disk IOPS`

- `Edit Tenant Kubernetes Policy`
- `Change Owner`
- `Create a Disk`
- `Delete a Disk`
- `Edit Disk Properties`
- `Move a Disk`
- `Create a Shared Disk`

In the “Provider VDC” → View section, enable `Limited Provider VDC Storage Policy View` and `Limited Provider VDC View`:



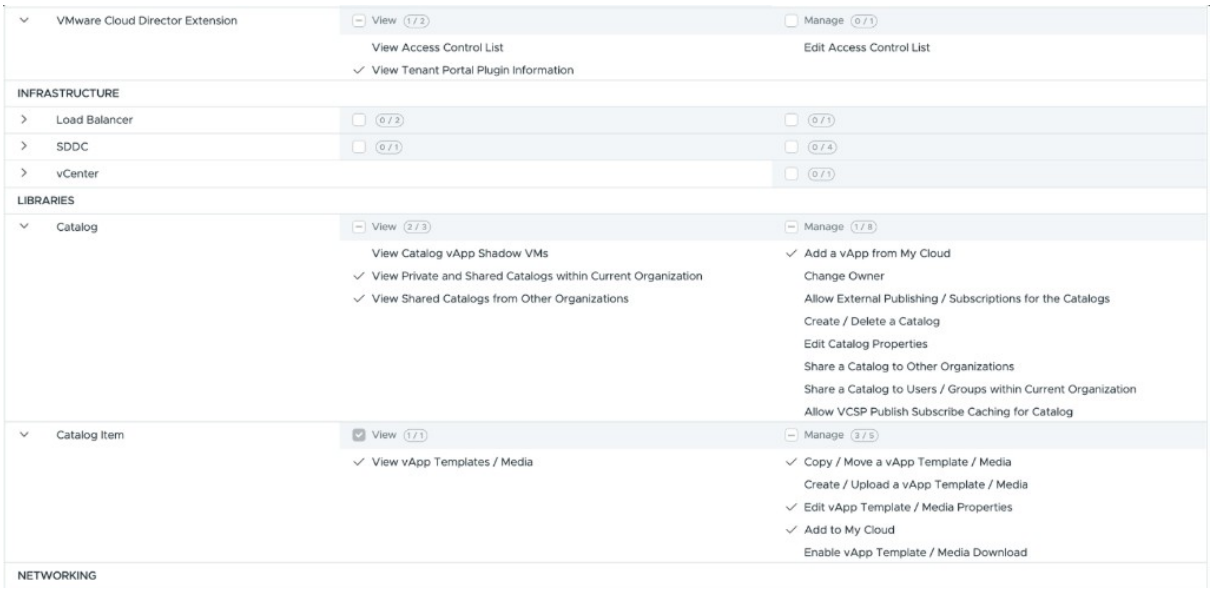
3. In the “COMPUTE” → “vApp” section, enable the rights as shown in the screenshot. Make sure that `Preserve All ExtraConfig Elements During OVF Import and Export` is enabled. In the verified role, View includes `View Encryption Status of VMs and VM's disks` and `View VM metrics`; `View vApp Shadow VMs` and the other Preserve ExtraConfig options are not enabled:



4. In the “LIBRARIES” section, enable the following rights:

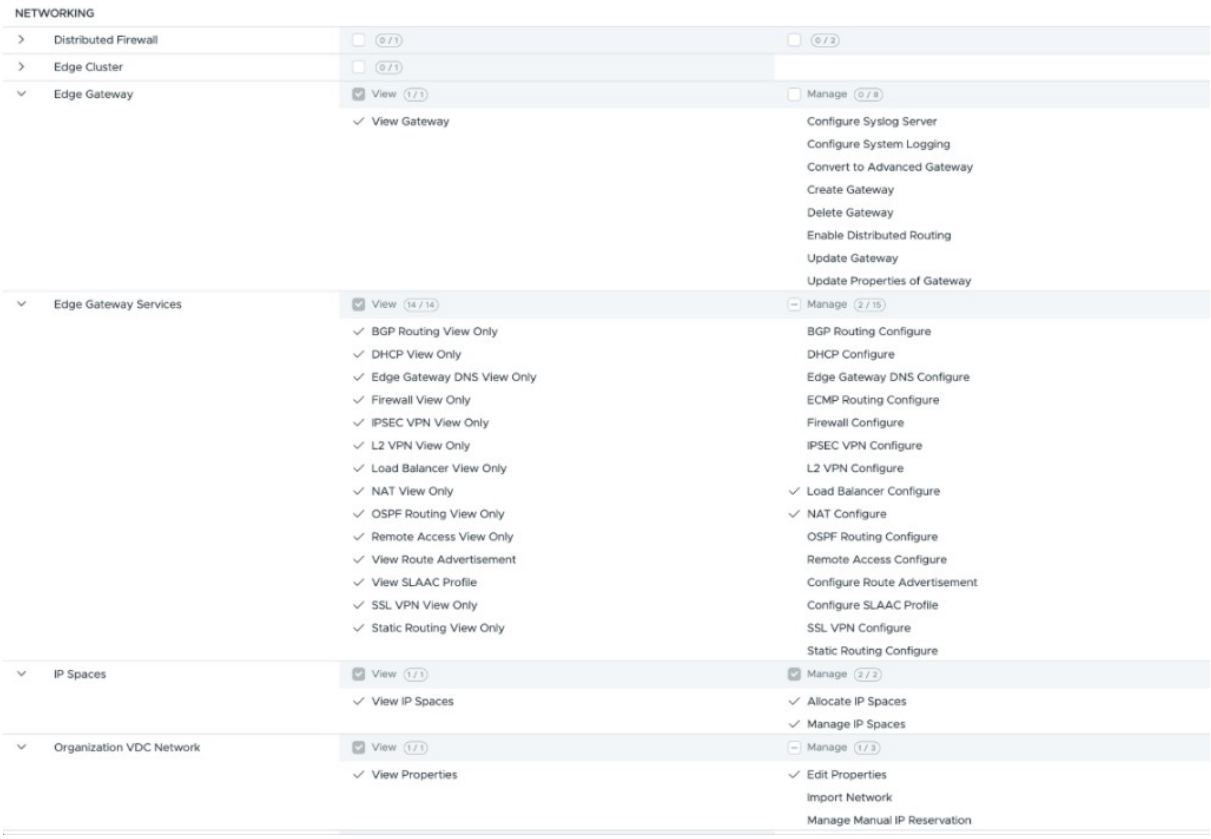
   - Catalog → View: `View Private and Shared Catalogs within Current Organization`, `View Shared Catalogs from Other Organizations` (to use images from shared catalogs)
   - Catalog → Manage: `Add a vApp from My Cloud`
   - Catalog Item → View: `View vApp Templates / Media`
   - Catalog Item → Manage: `Copy / Move a vApp Template / Media`, `Edit vApp Template / Media Properties`, `Add to My Cloud`

   In the “VMware Cloud Director Extension” → View section, enable `View Tenant Portal Plugin Information`:

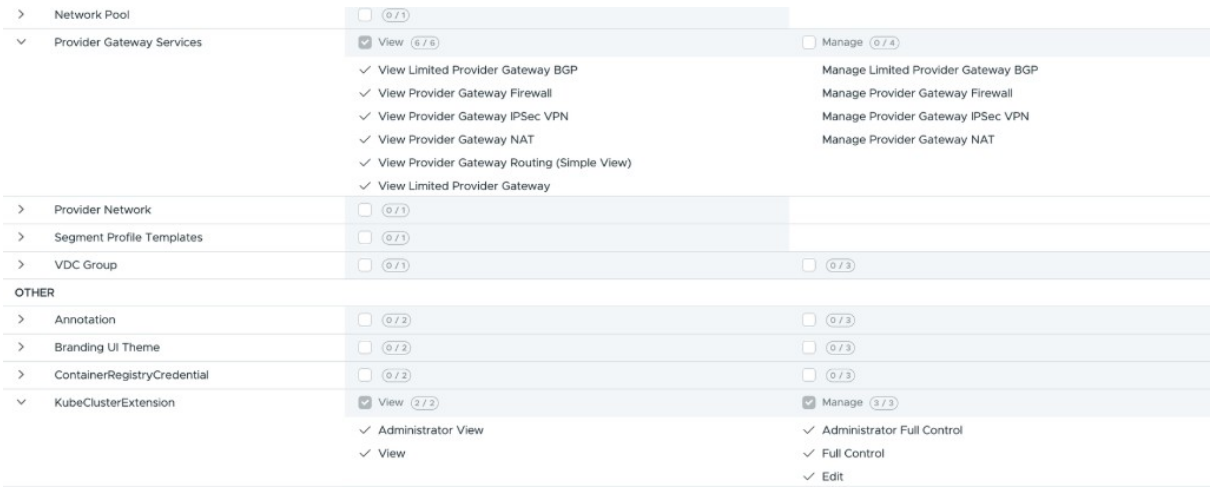


5. In the “NETWORKING” section, enable the following rights:

   - Edge Gateway → View: `View Gateway`
   - Edge Gateway Services → View: all items
   - Edge Gateway Services → Manage: `Load Balancer Configure`, `NAT Configure`
   - IP Spaces → View: `View IP Spaces`
   - IP Spaces → Manage: `Allocate IP Spaces`, `Manage IP Spaces`
   - Organization VDC Network → View: `View Properties`
   - Organization VDC Network → Manage: `Edit Properties` (required to create networks in the organization)



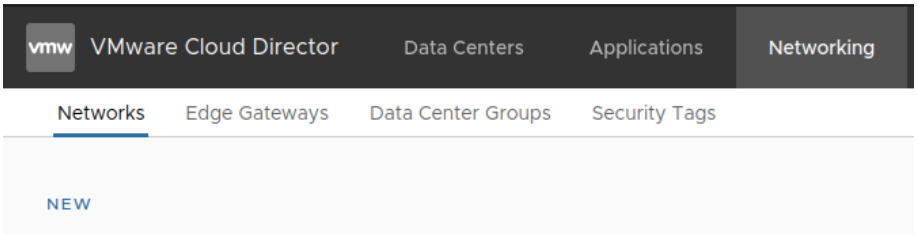
6. In the “Provider Gateway Services” section, enable all View rights. In the “KubeClusterExtension” section, enable all View and Manage rights:



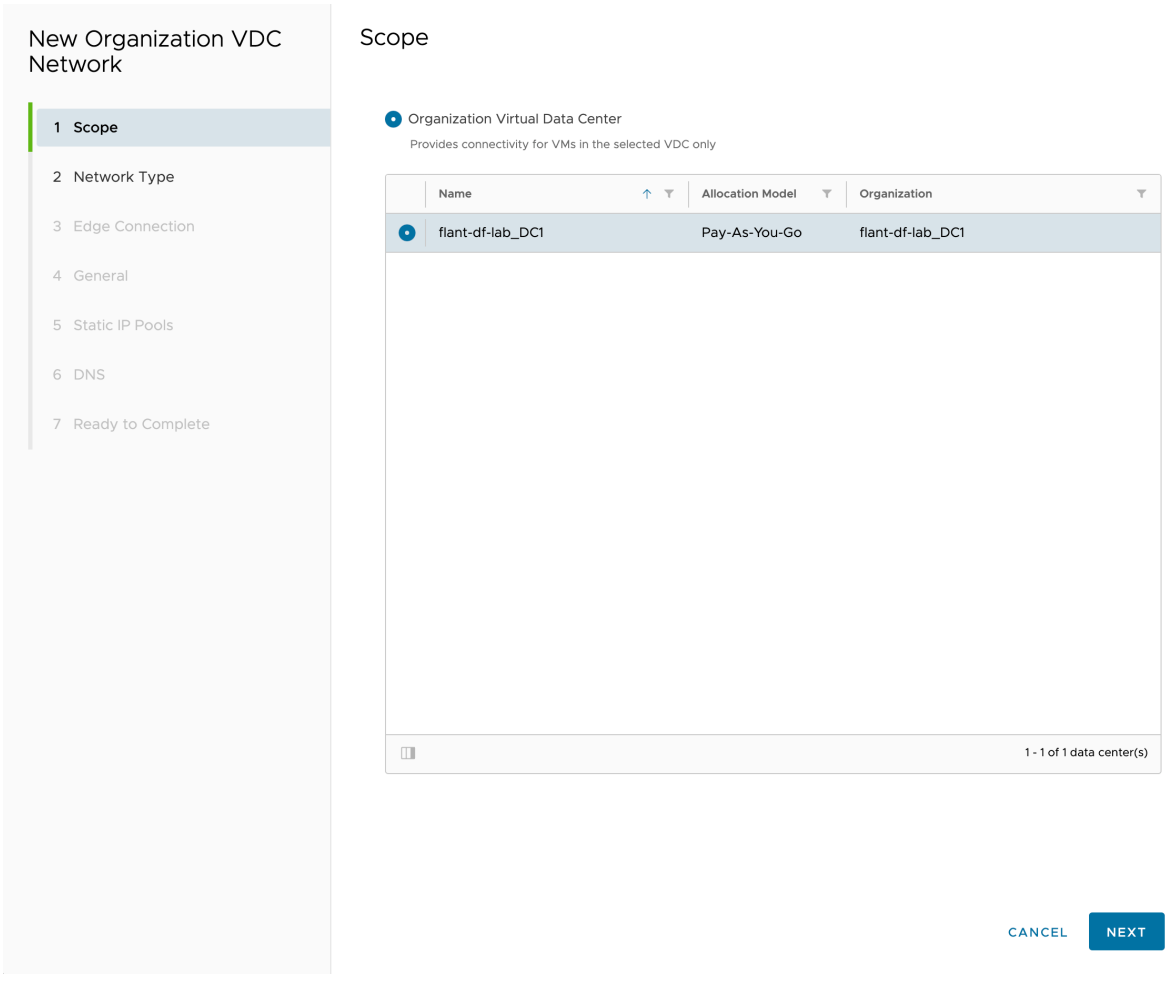
Adding a network

> **i** This instruction is applicable only for the `Standard` layout.

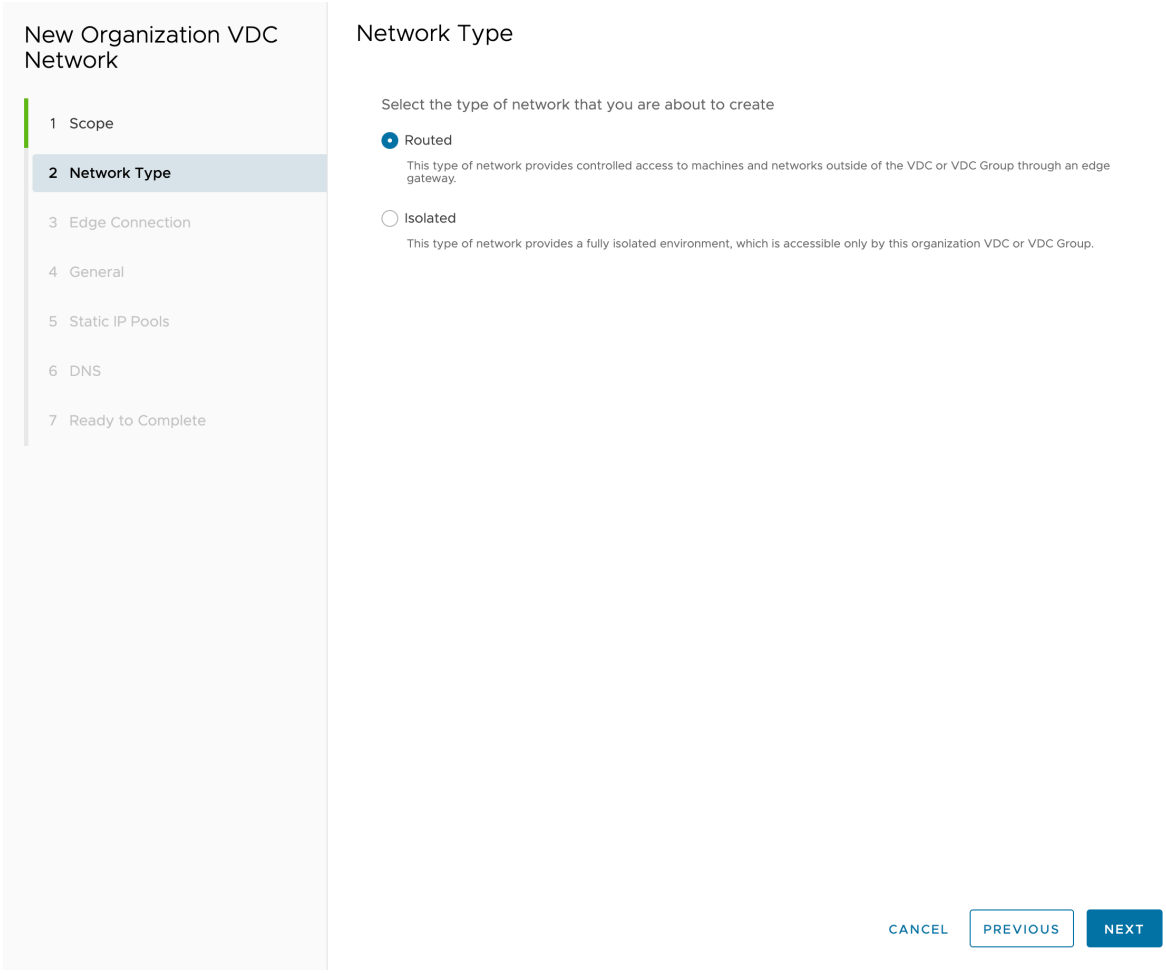
1. Go to the “Networking” tab and click “NEW”:



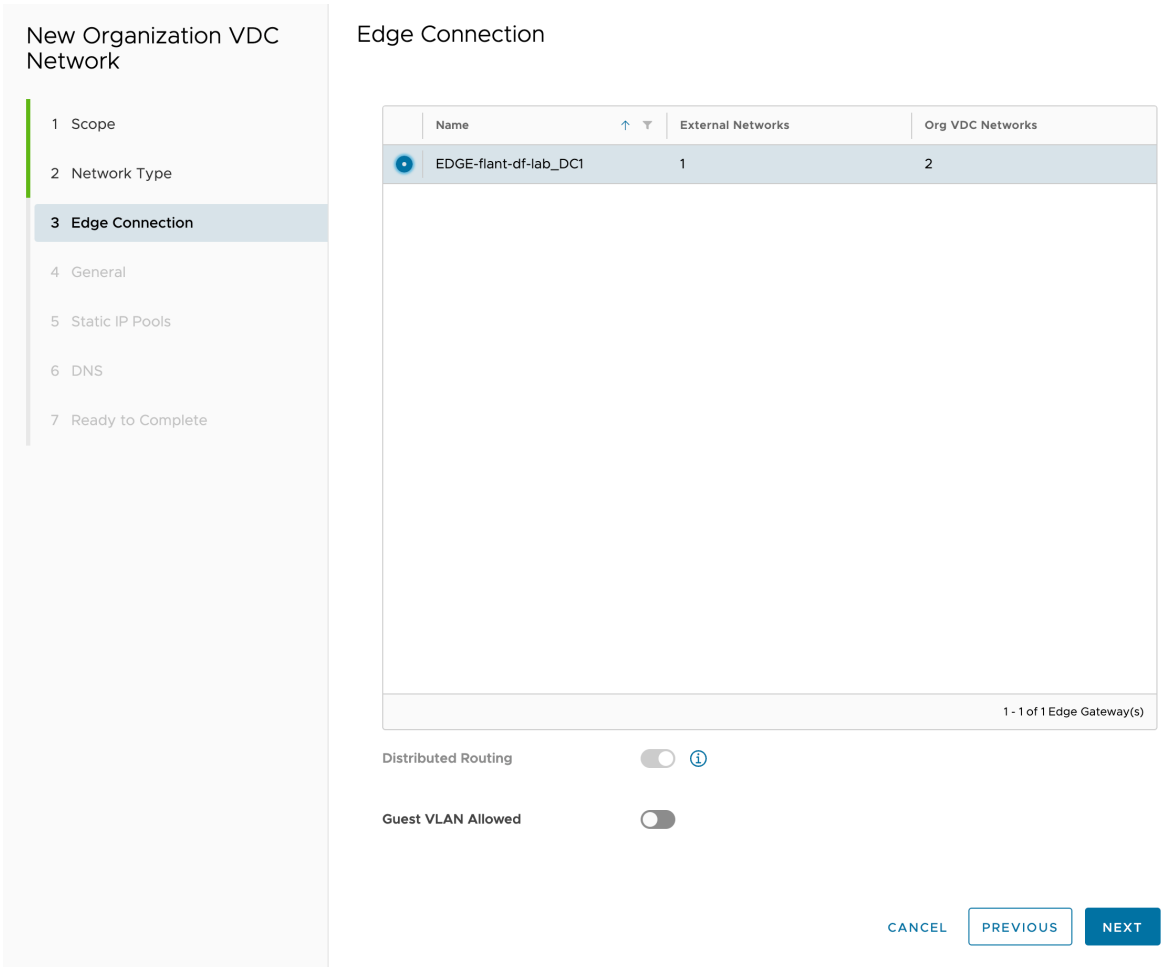
2. Select the Data Center:



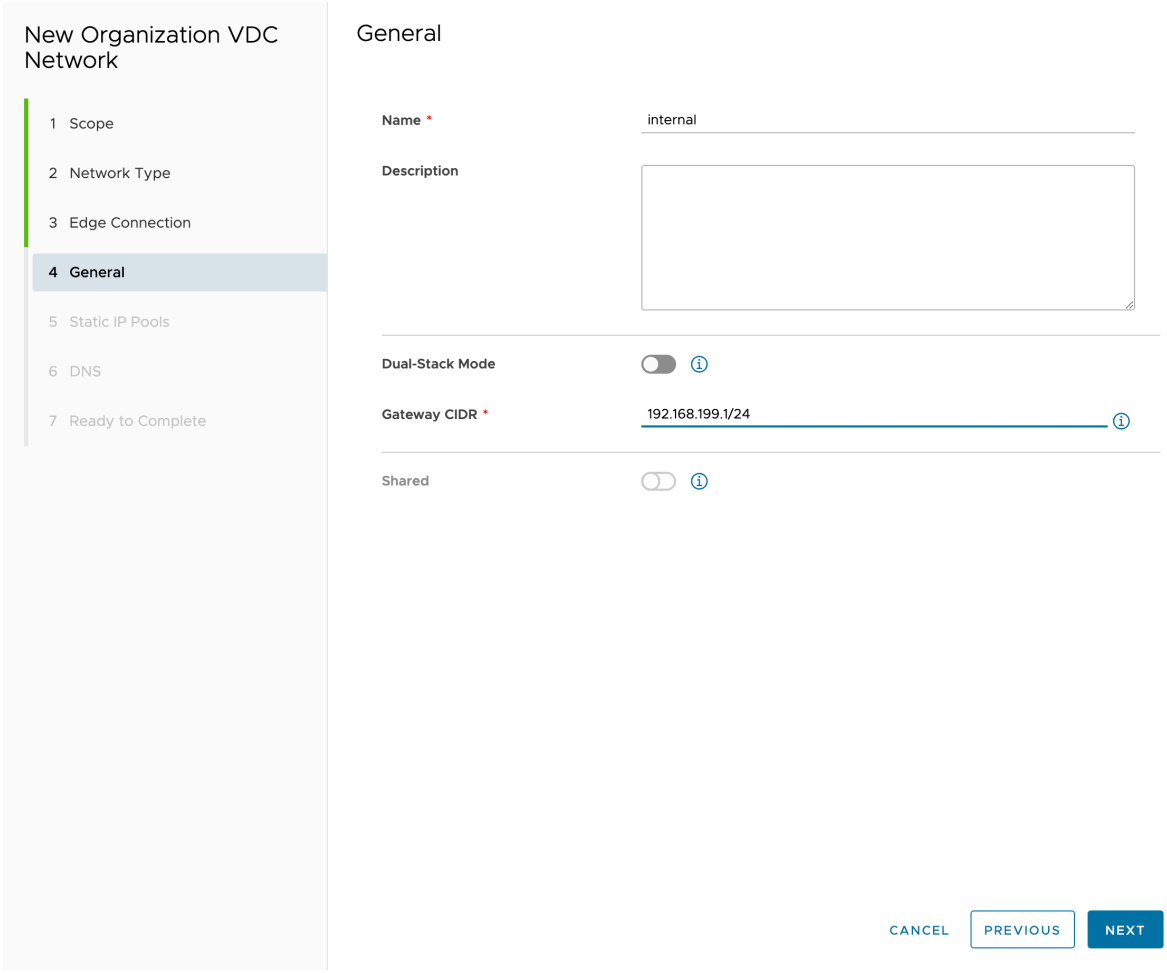
3. At the “Network type” step, select “Routed”:



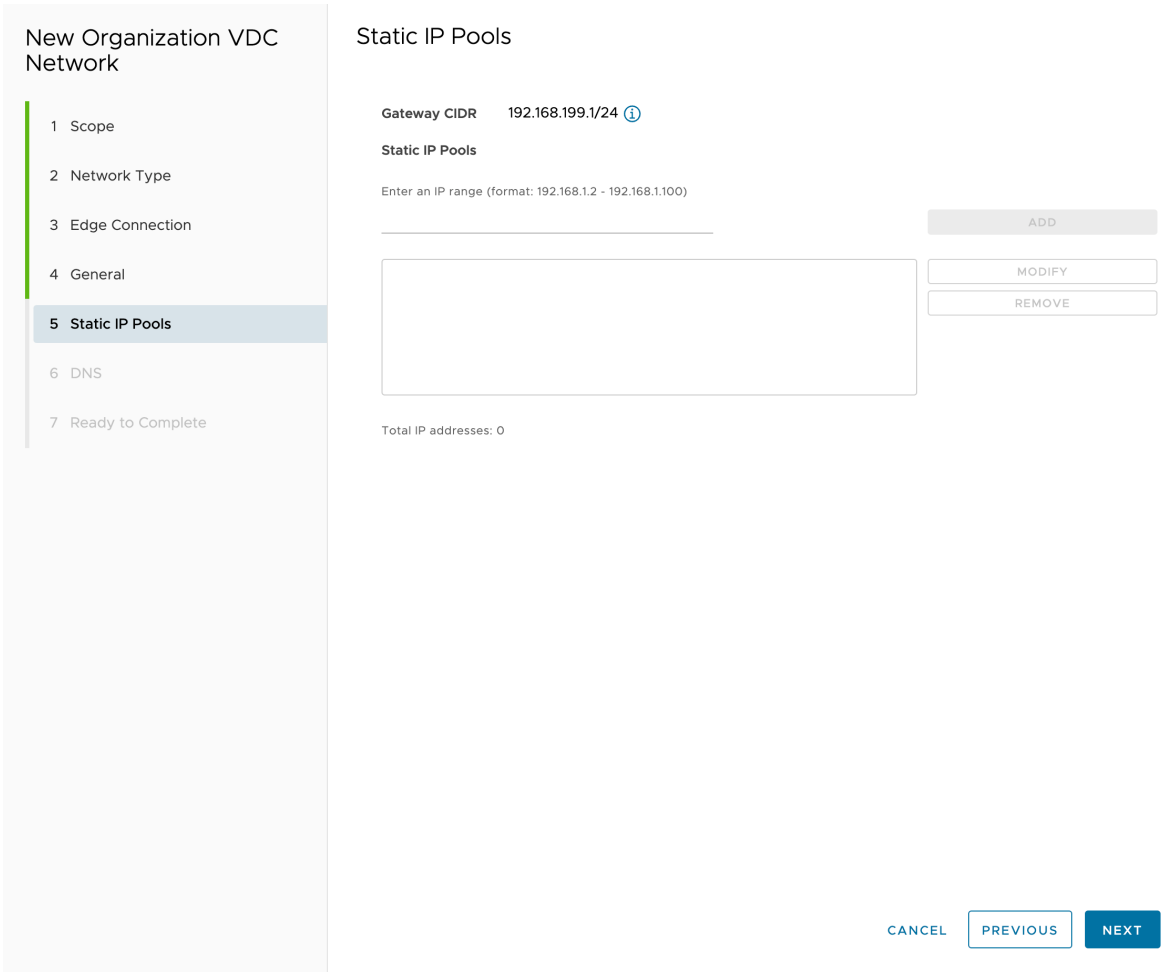
4. Connect `EdgeRouter` to the network:



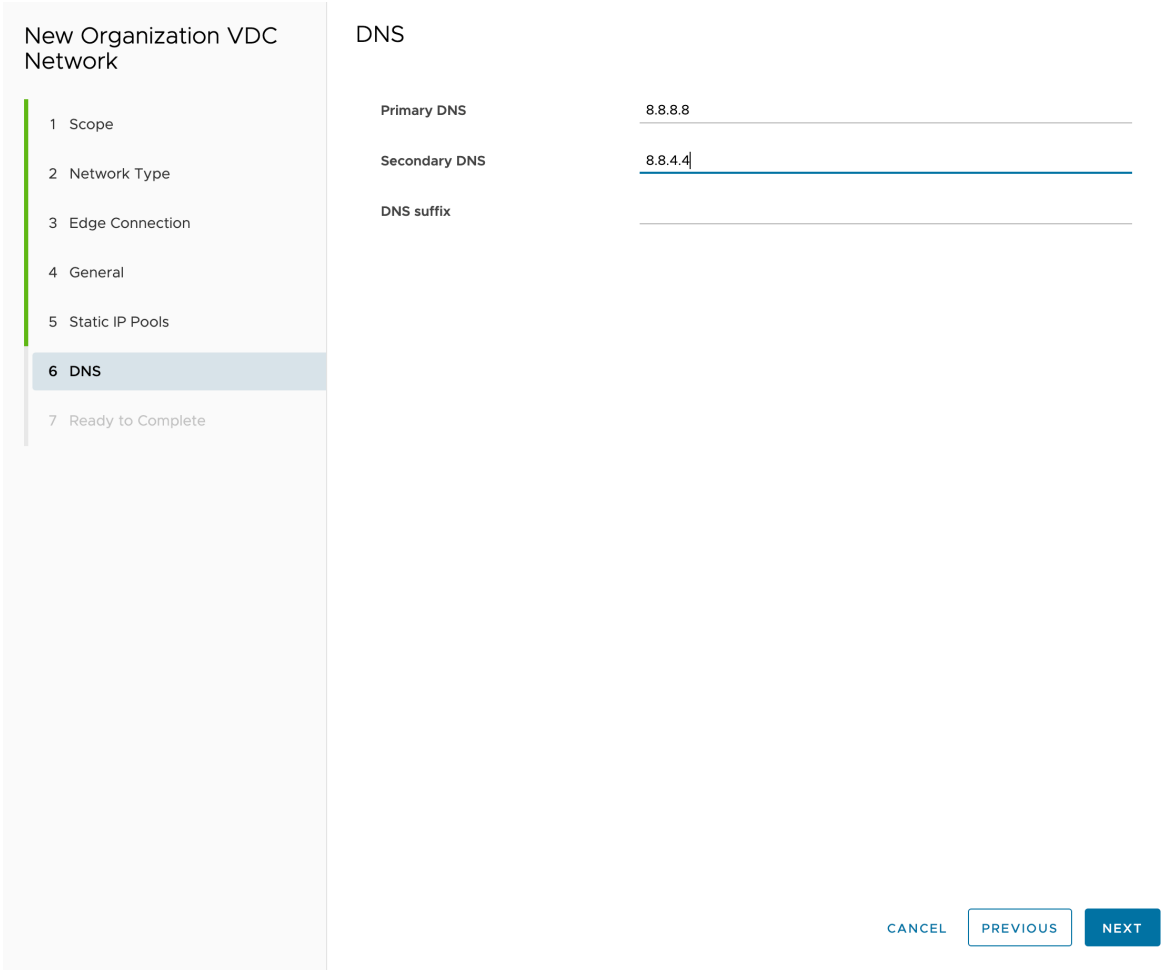
5. Specify the network name and CIDR:



6. Do not add “Static IP Pools” because DHCP will be used:



7. Specify the DNS server addresses:



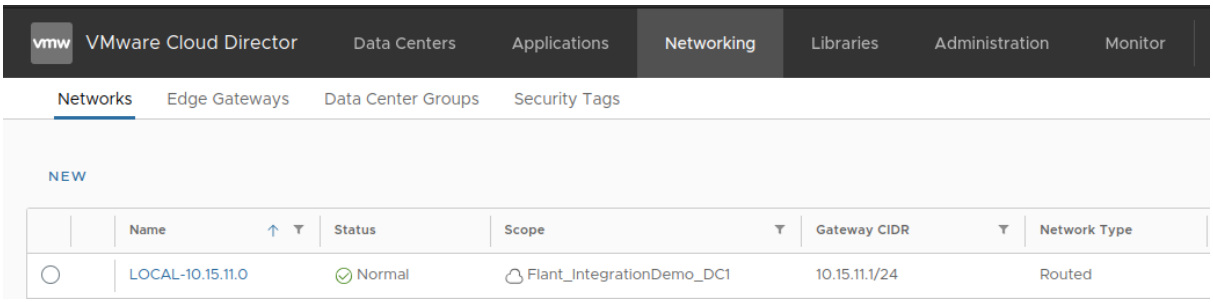
Configuring DHCP

> **i** This instruction is applicable only for the `Standard` layout.

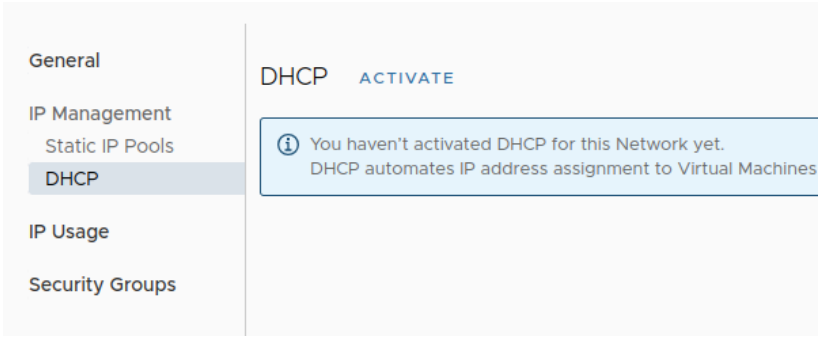
To provision nodes dynamically, enable the DHCP server for the internal network.

> **i** We recommend allocating the beginning of the network address range to system consumers (control plane, frontend nodes, system nodes) and the rest to the DHCP pool. For example, for a `/24` mask network it would be enough to allocate 20 addresses to system consumers.

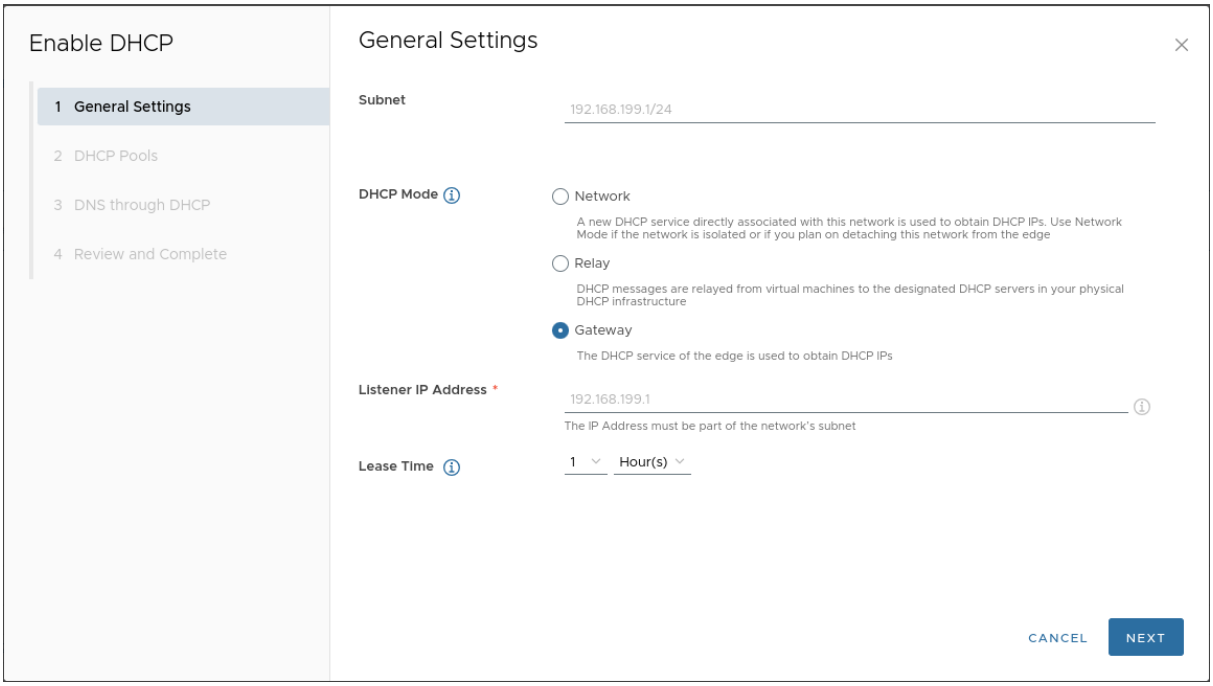
1. Click the “Networking” tab and open the network you created:



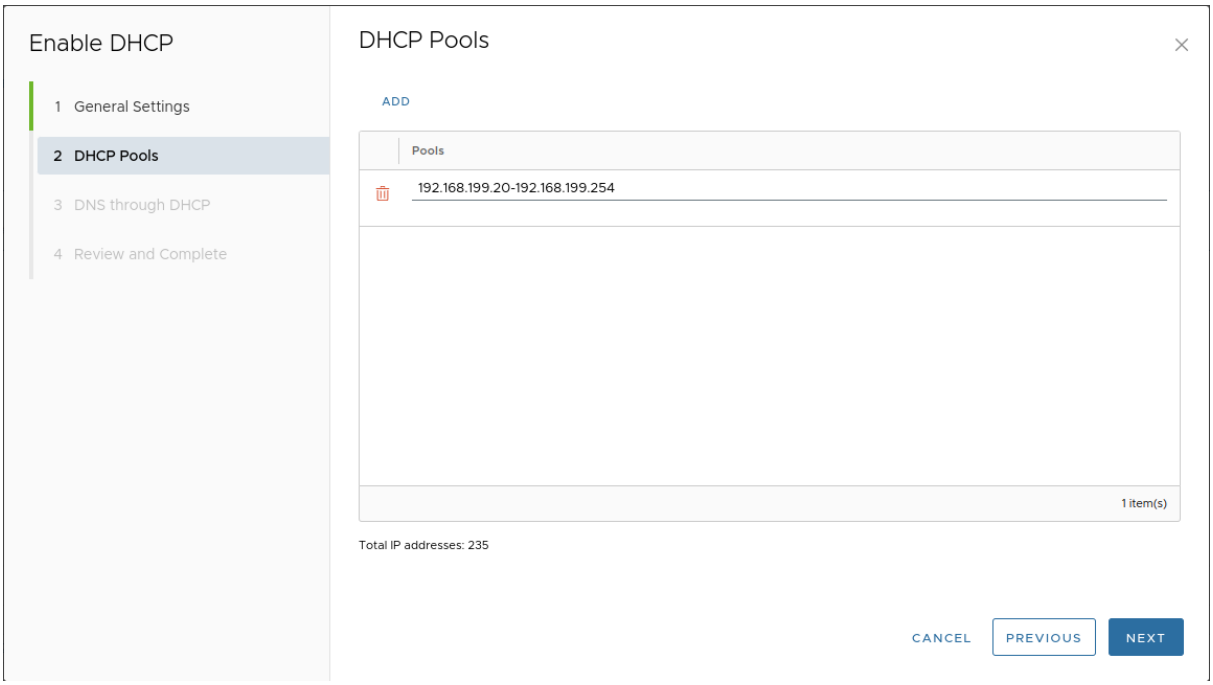
2. In the opened window, select “IP Management” → “DHCP” → “Activate”:



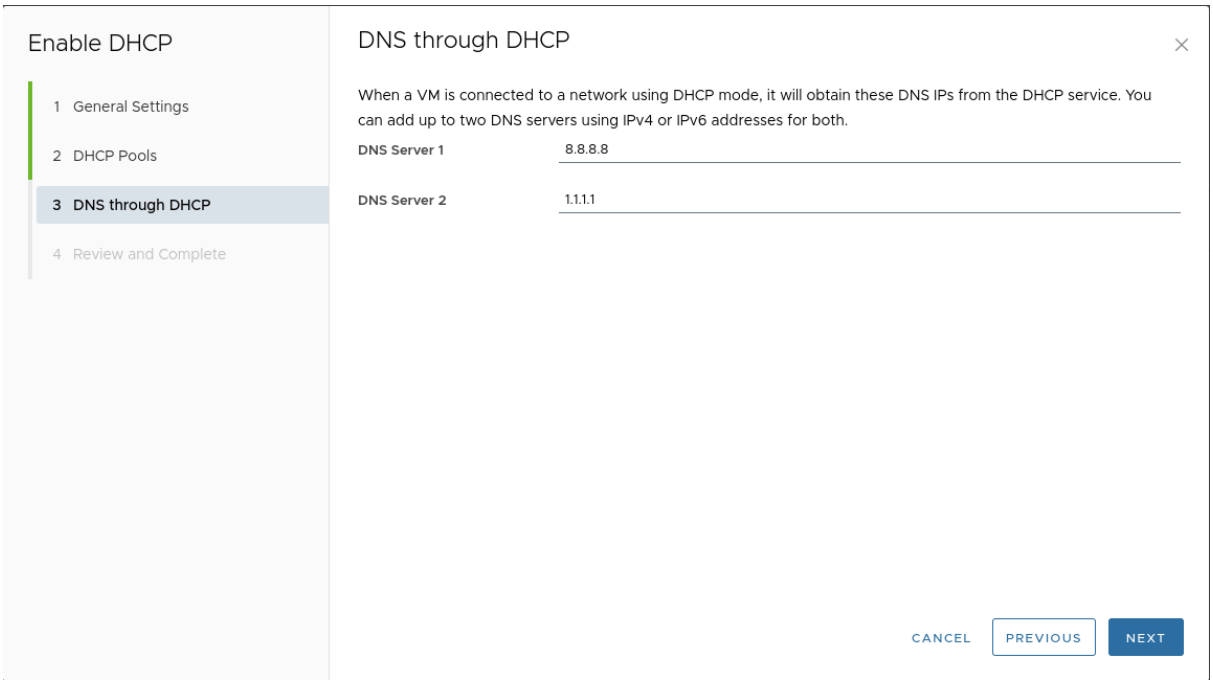
3. In the “General settings” tab, set the parameters as shown in the example:



4. Add a pool:



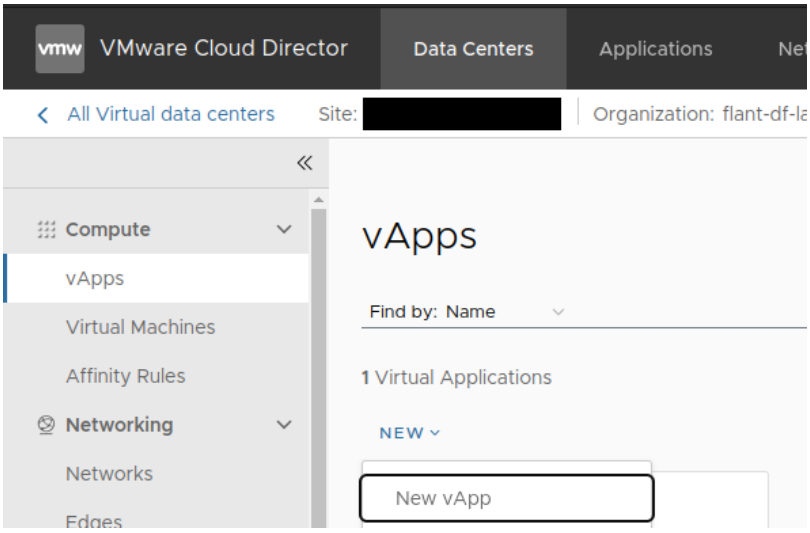
5. Set the DNS server addresses:



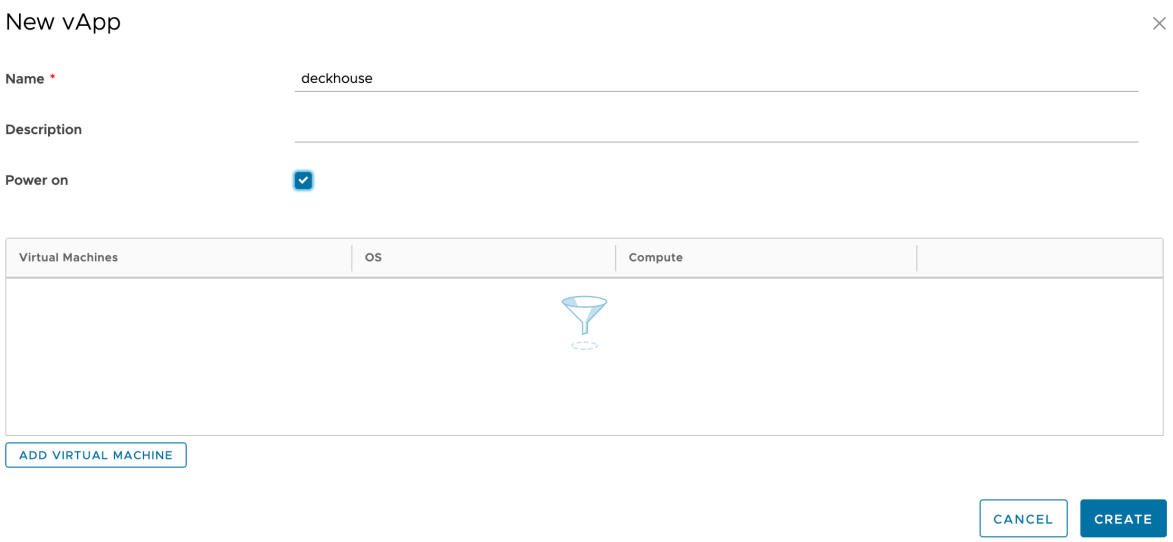
Adding a vApp

> **i** This instruction is applicable only for the `Standard` deployment layout.

1. Switch to the “Data Centers” tab → “vApps” → “NEW” → “New vApp”:



2. Specify a name and enable the vApp:

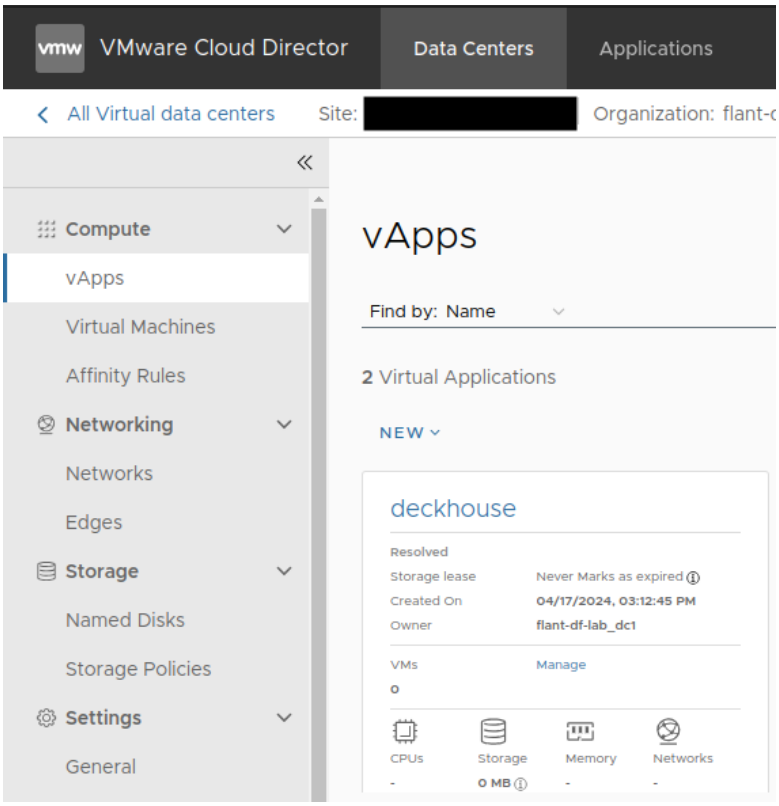


Adding a network to the vApp

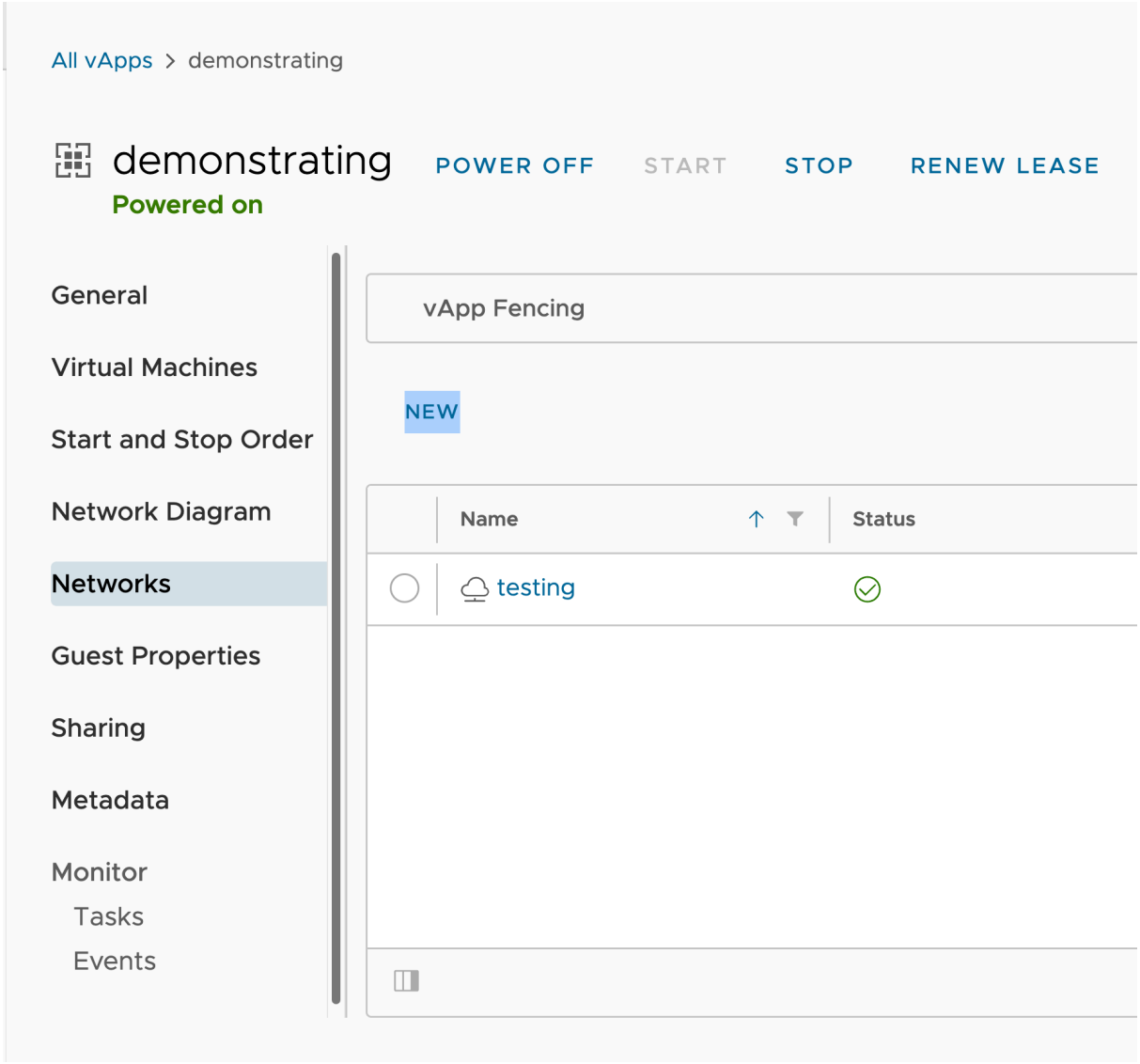
> **i** This instruction is applicable only for the `Standard` deployment layout.

Once the vApp is created, connect the created internal network to it.

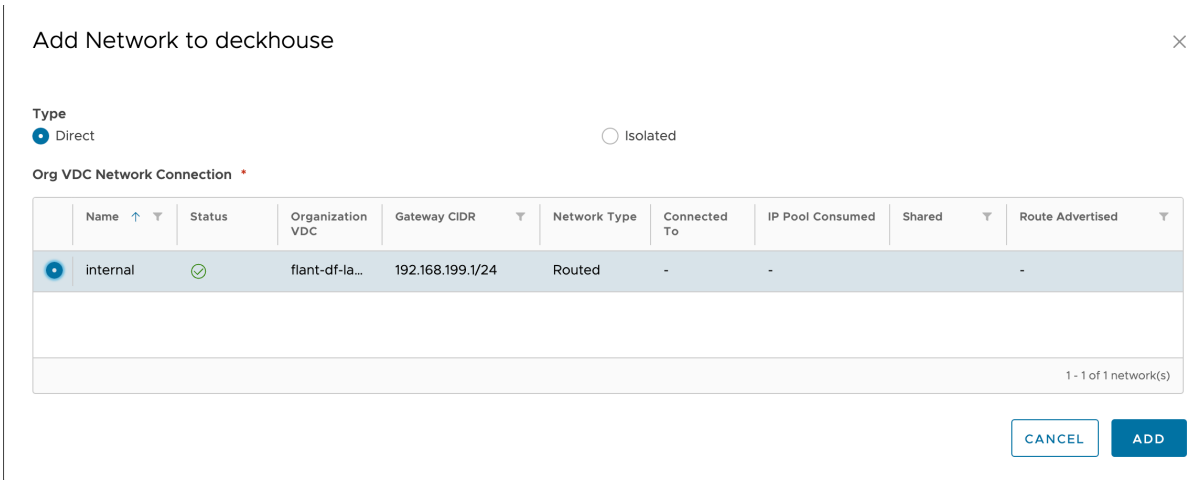
1. Switch to the “Data Centers” tab → “vApps” and open the target vApp:



2. Go to the “Networks” tab and click “NEW”:



3. In the opened window, click the “Direct” type and select the network:



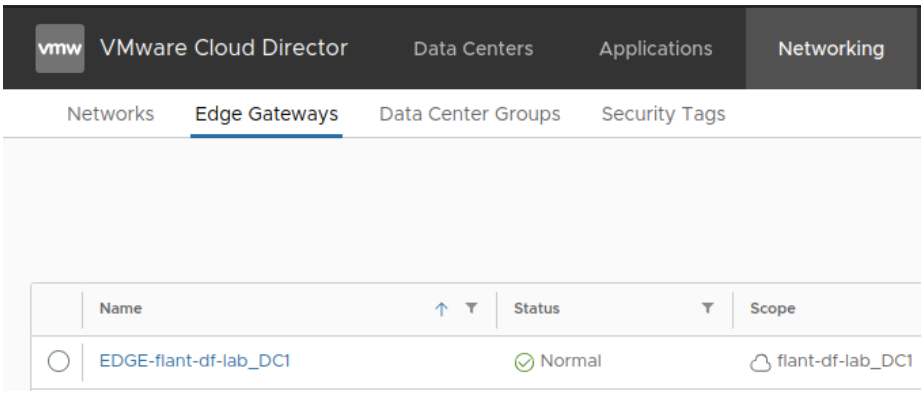
Incoming traffic

Incoming traffic should be routed to the edge router (ports `80`, `443`) using DNAT rules to be forwarded to a dedicated address on the internal network.
This address can be created by running MetalLB in L2 mode for dedicated frontend nodes.

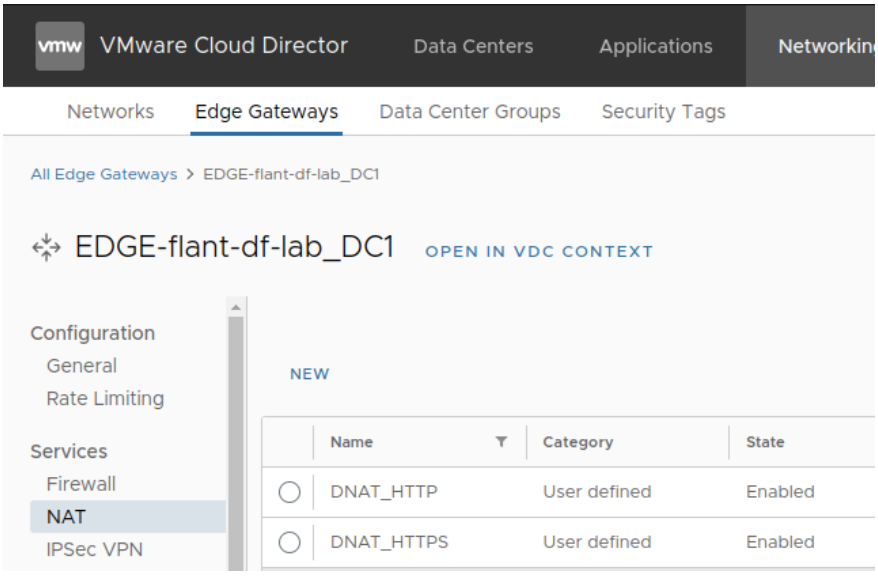
Configuring DNAT/SNAT rules on the edge gateway

> **i** This instruction is applicable only for the `Standard` deployment layout.

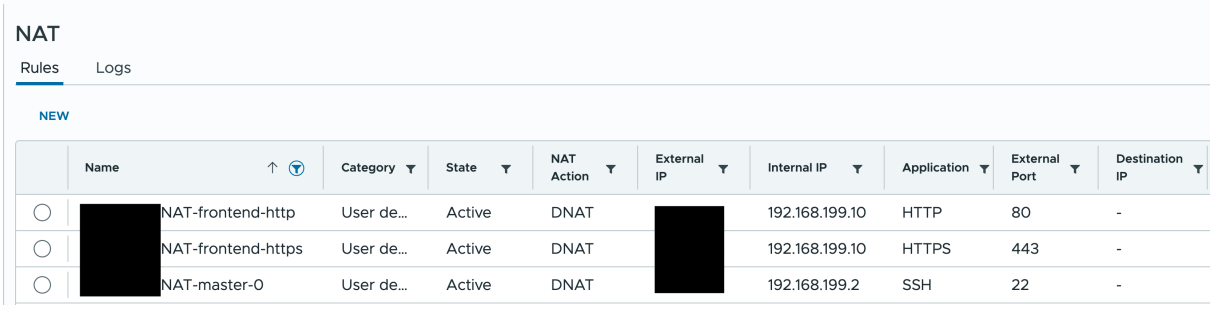
1. Navigate to the “Networking” tab → “Edge Gateways” and open the edge gateway:



2. Switch to the “Services” tab → “NAT”:



3. Add the following rules:



   The first two rules are used for incoming traffic, while the third rule is used for SSH access to the control plane host (without this rule the installation will not be possible).

4. To allow virtual machines to access the Internet, configure SNAT rules following the example:



   This rule will allow virtual machines from the `192.168.199.0/24` subnet to access the Internet.

Let’s Encrypt certificate issuance

> **i** This instruction is applicable only for the `Standard` deployment layout.

When using `cert-manager` with the HTTP-01 ACME challenge type, you may need to configure Hairpin NAT (NAT loopback) on the Edge Gateway.

This is required when the Ingress domain name resolves to the external IP address of the Edge Gateway, while requests to

that address are made from **inside** the cluster (from the node network). In this case, the traffic must be correctly routed back to the internal network to the Ingress address (for example, the MetalLB address).

This is important for issuing Let's Encrypt certificates because cert-manager performs a preliminary self-check of the challenge URL before sending a request to the ACME provider. If the URL is not reachable from inside the cluster, certificate issuance will not start, even if the address is accessible from outside.

Example (cluster network `192.168.199.0/24`):

- internal node network: `192.168.199.0/24`
- external Edge Gateway IP address: `194.117.83.19`
- internal Ingress address (for example, MetalLB): `192.168.199.251`.

In this case, you must configure Hairpin NAT for traffic from the `192.168.199.0/24` network that is sent to the external IP `194.117.83.19`, with translation to the internal Ingress address `192.168.199.251`.

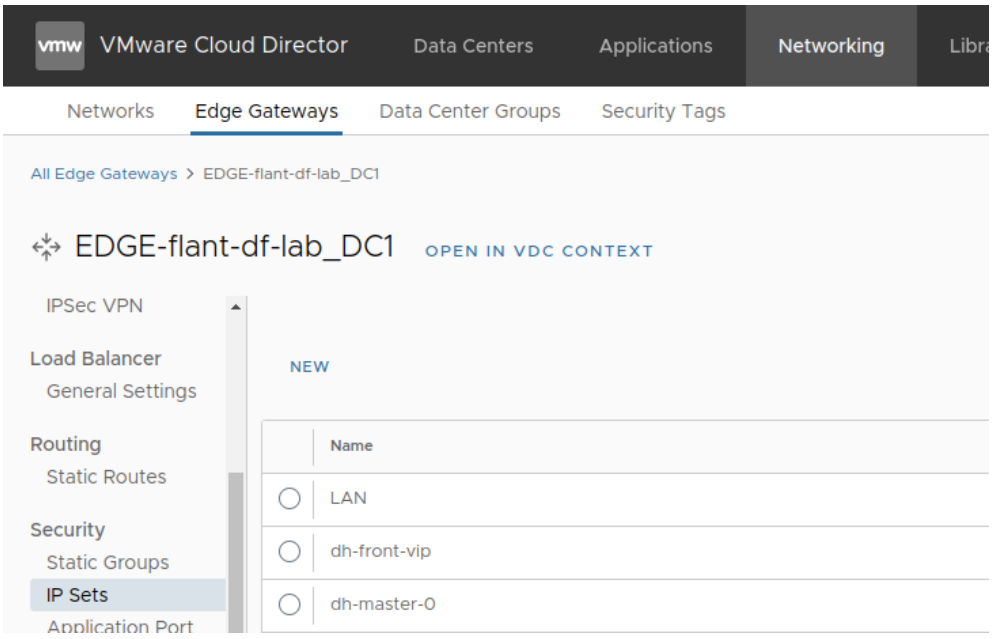
> **i**
>
> Hairpin NAT configuration may be required for Edge Gateway based on NSX-V.
>
> For NSX-T, the loopback scenario is often supported by default, but the actual behavior depends on the VMware Cloud Director provider configuration.

Configuring a firewall

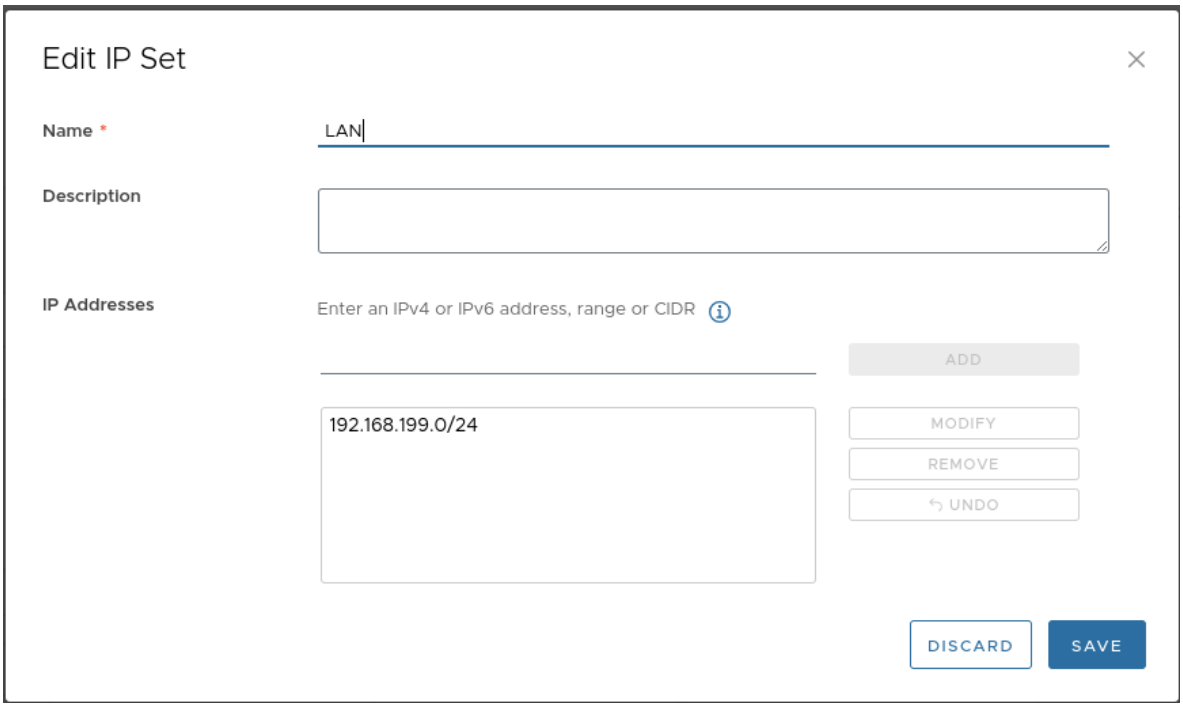
> **i**
>
> This instruction is applicable only for the `Standard` deployment layout.

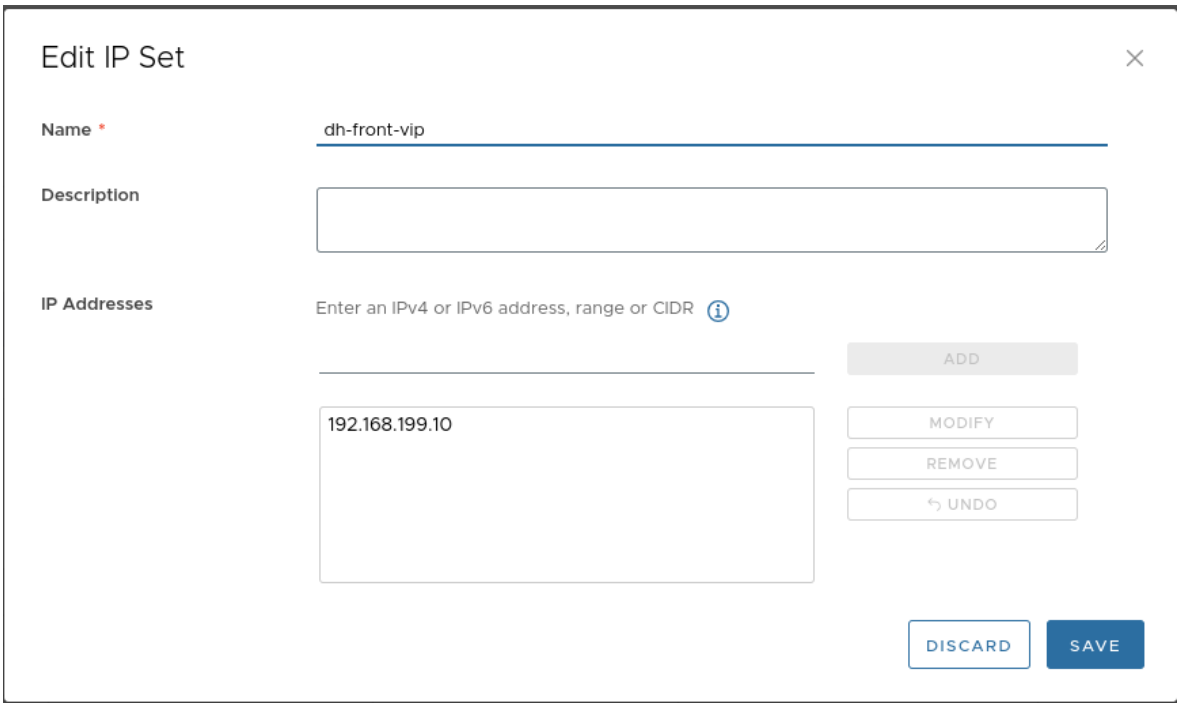
Once DNAT is configured, set up the firewall. Start by configuring the IP sets.

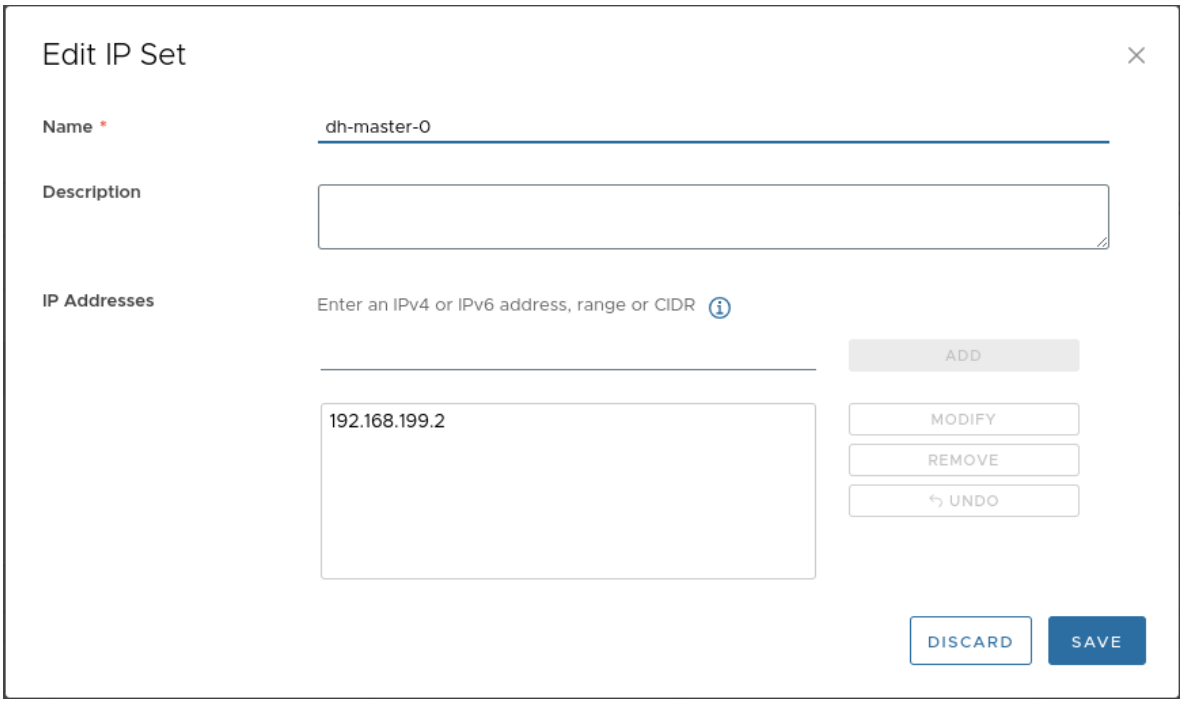
1. Switch to the "Security" tab → "IP Sets":



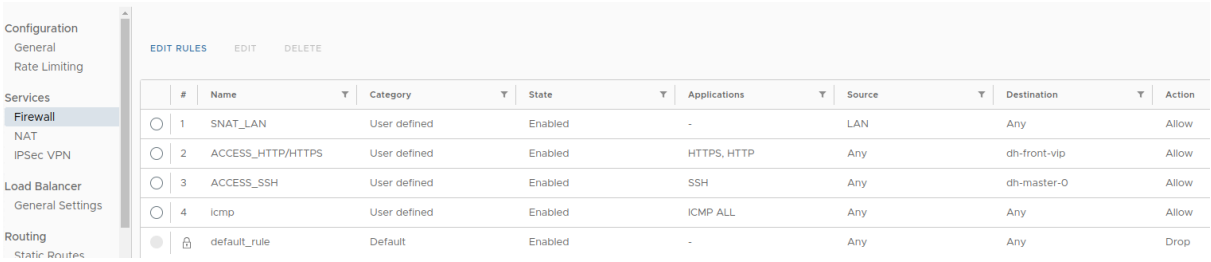
2. Create the following set of IPs (the MetalLB address here is `.10` and the control plane node address is `.2`):


Edit IP Set
Name *
LAN
Description
IP Addresses
Enter an IPv4 or IPv6 address, range or CIDR
ADD
192.168.199.0/24
MODIFY
REMOVE
UNDO
DISCARD
SAVE


Edit IP Set
Name *
dh-front-vip
Description
IP Addresses
Enter an IPv4 or IPv6 address, range or CIDR
ADD
192.168.199.10
MODIFY
REMOVE
UNDO
DISCARD
SAVE



3. Add the following firewall rules:



Virtual machine template

⚠ The provider is confirmed to work with Ubuntu 22.04-based virtual machine templates only.

⚠ Disable vApp/Guest OS Customization (Guest Customization, vApp Customization, and similar mechanisms) for the cluster template and virtual machines in VMware Cloud Director. DKP performs the initial node provisioning via `cloud-init` (OVF/VMware GuestInfo datasource). If customization is enabled, it may conflict with `cloud-init` and result in incorrect node initialization.

⚠ To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:

- Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- `CgroupsV2` support.
- Systemd version `244` or newer.
- `erofs` kernel module support.

For more information, see the documentation.

⚠ The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:

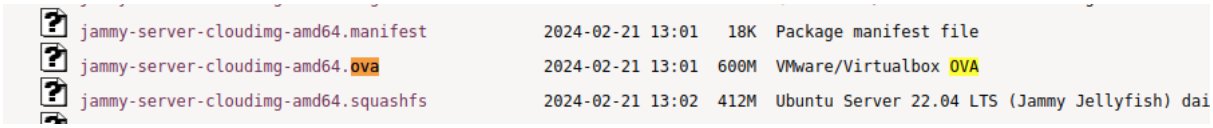
- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

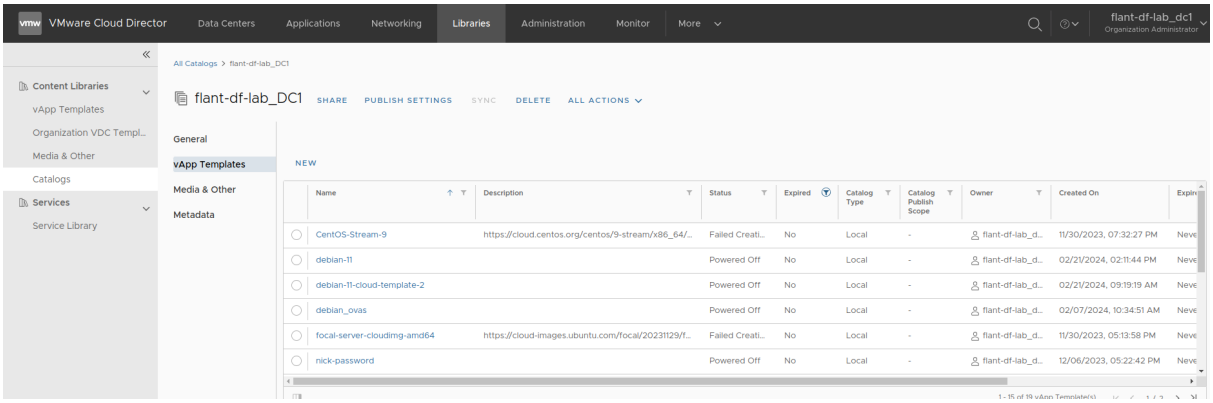
The example below uses the OVA file provided by Ubuntu, updated to include two fixes. Those fixes are essential for CloudPermanent nodes to be provisioned correctly and to be able to mount CSI-created disks.

Making a template from an OVA file

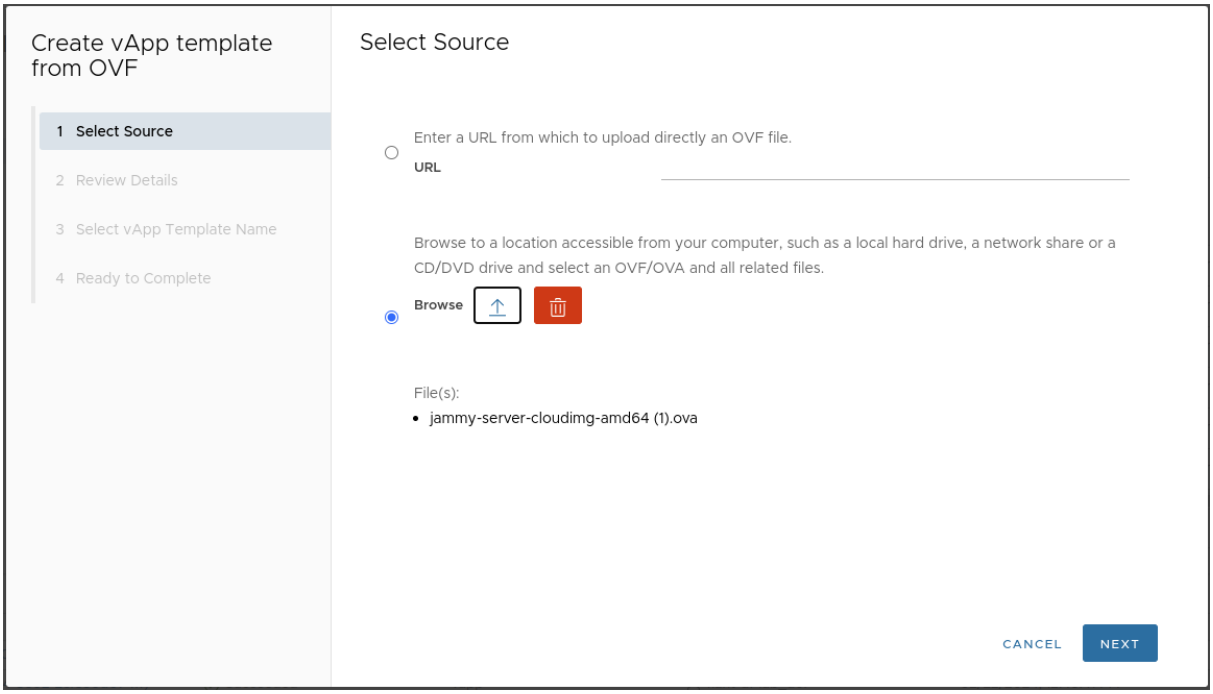
1. Download the OVA file:

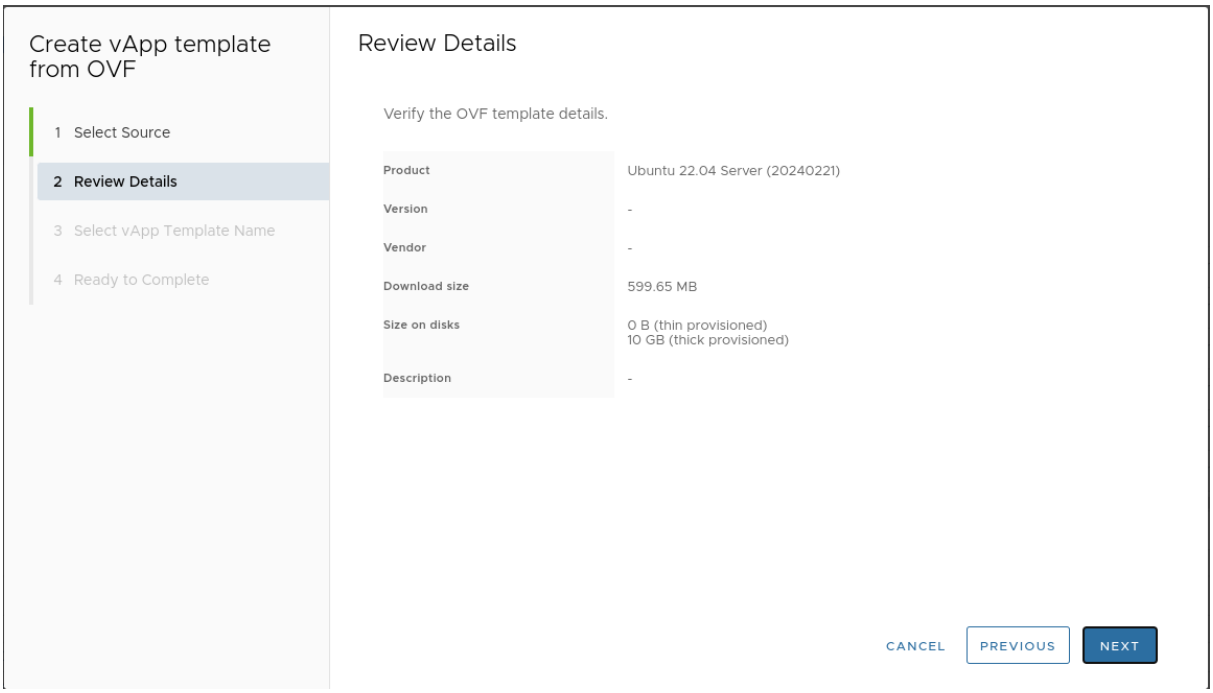


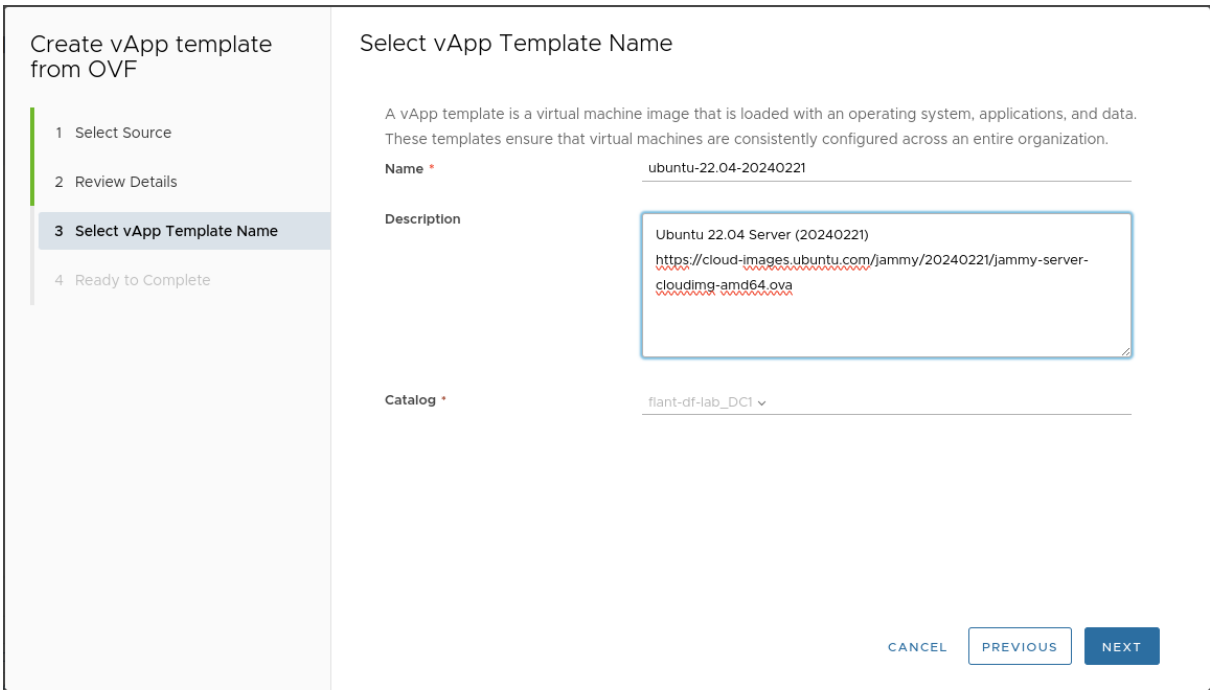
2. Switch to the “Libraries” tab → “Catalogs” → “Organization Catalog”:



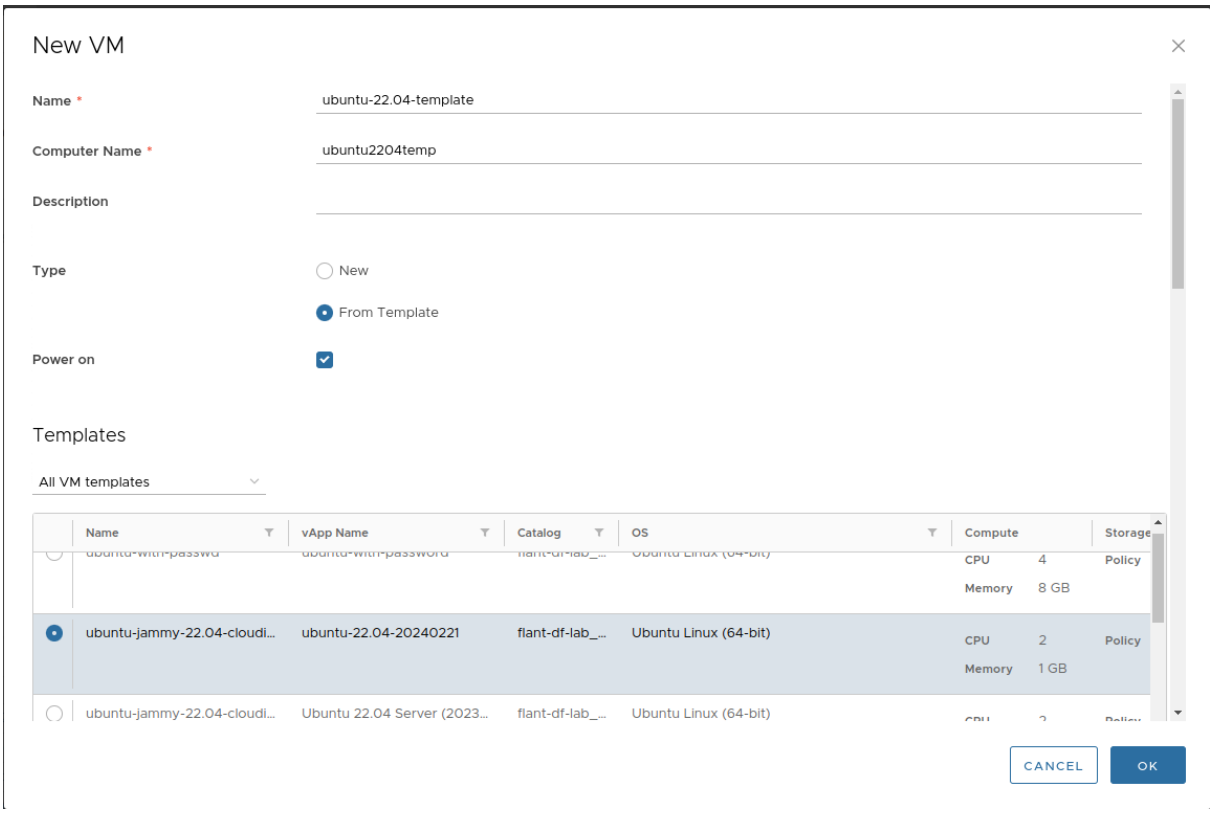
3. Select the template you downloaded and add it to the catalog:

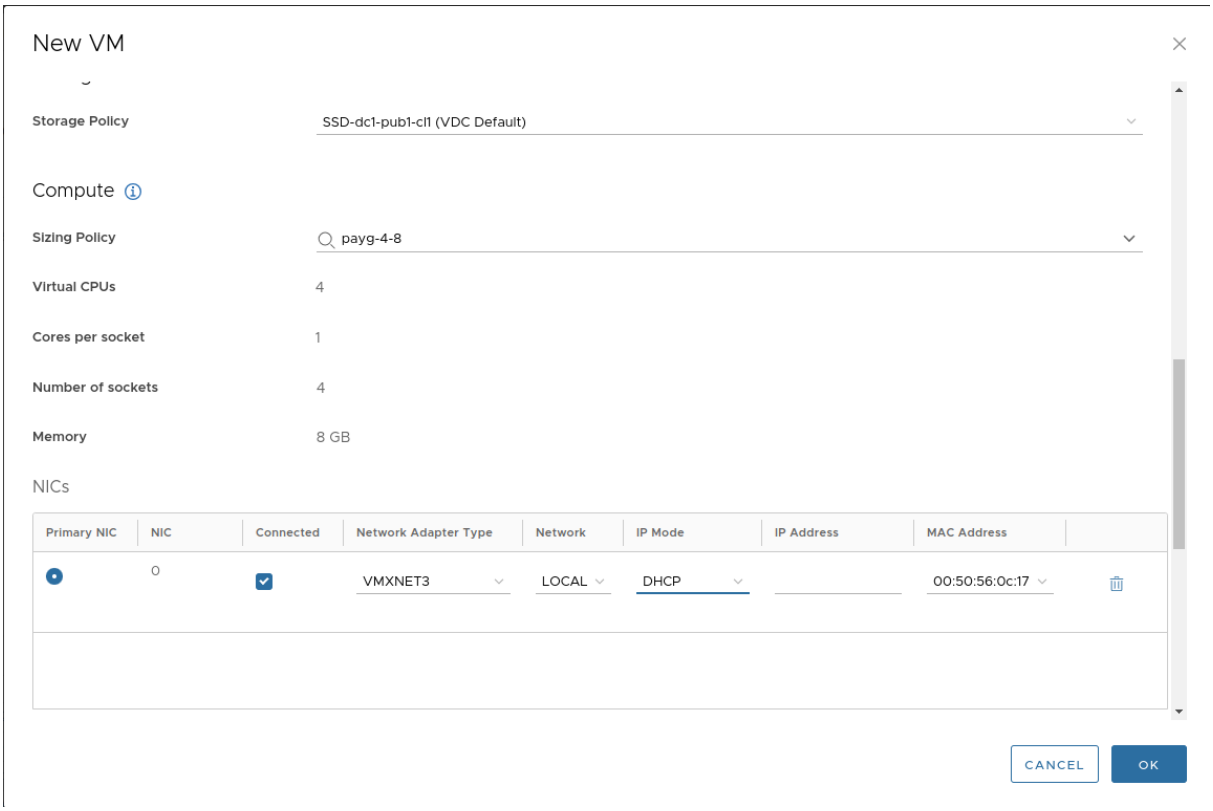




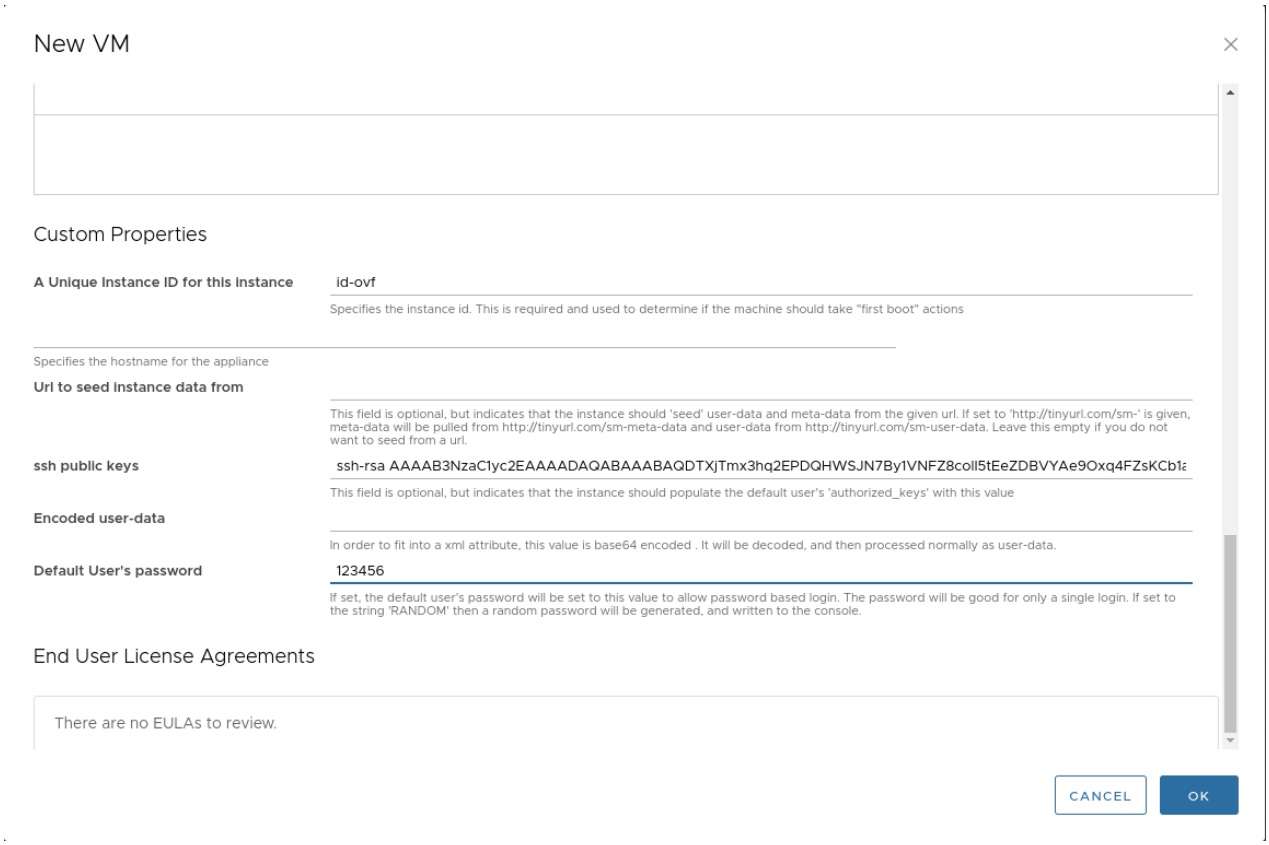


4. Create a virtual machine from the template:



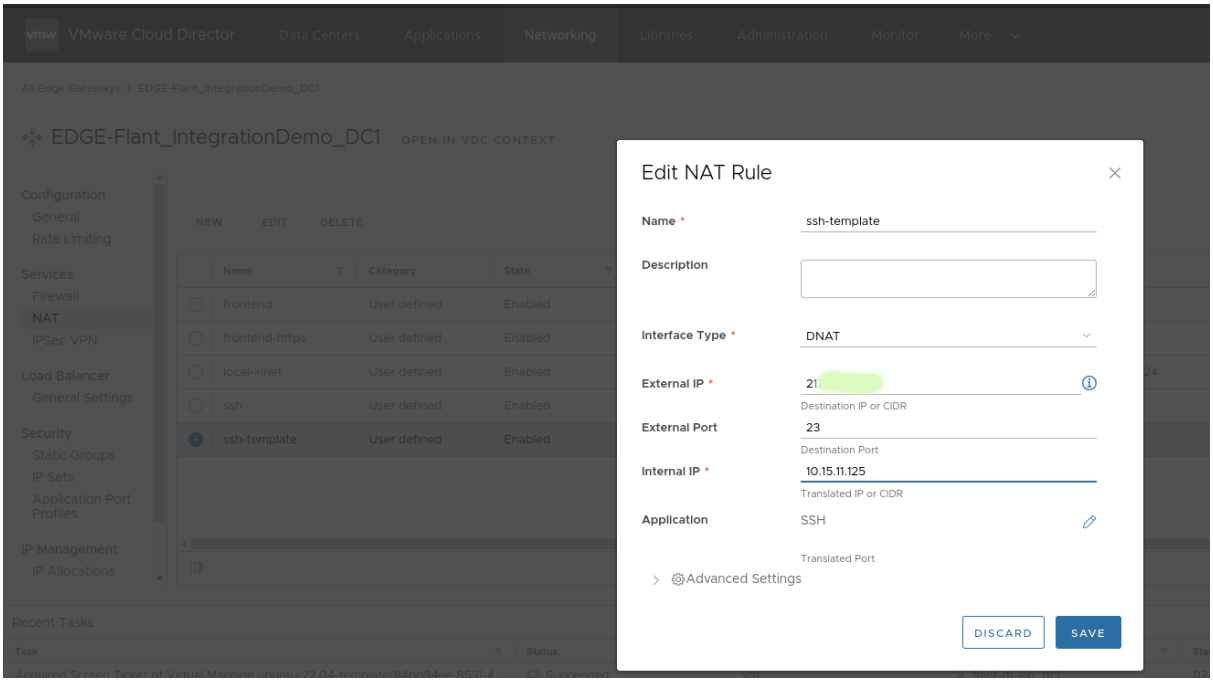


⚠ Enter the default password and public key. You will need them to log in to the VM console.



Follow these steps to be able to connect to the virtual machine:

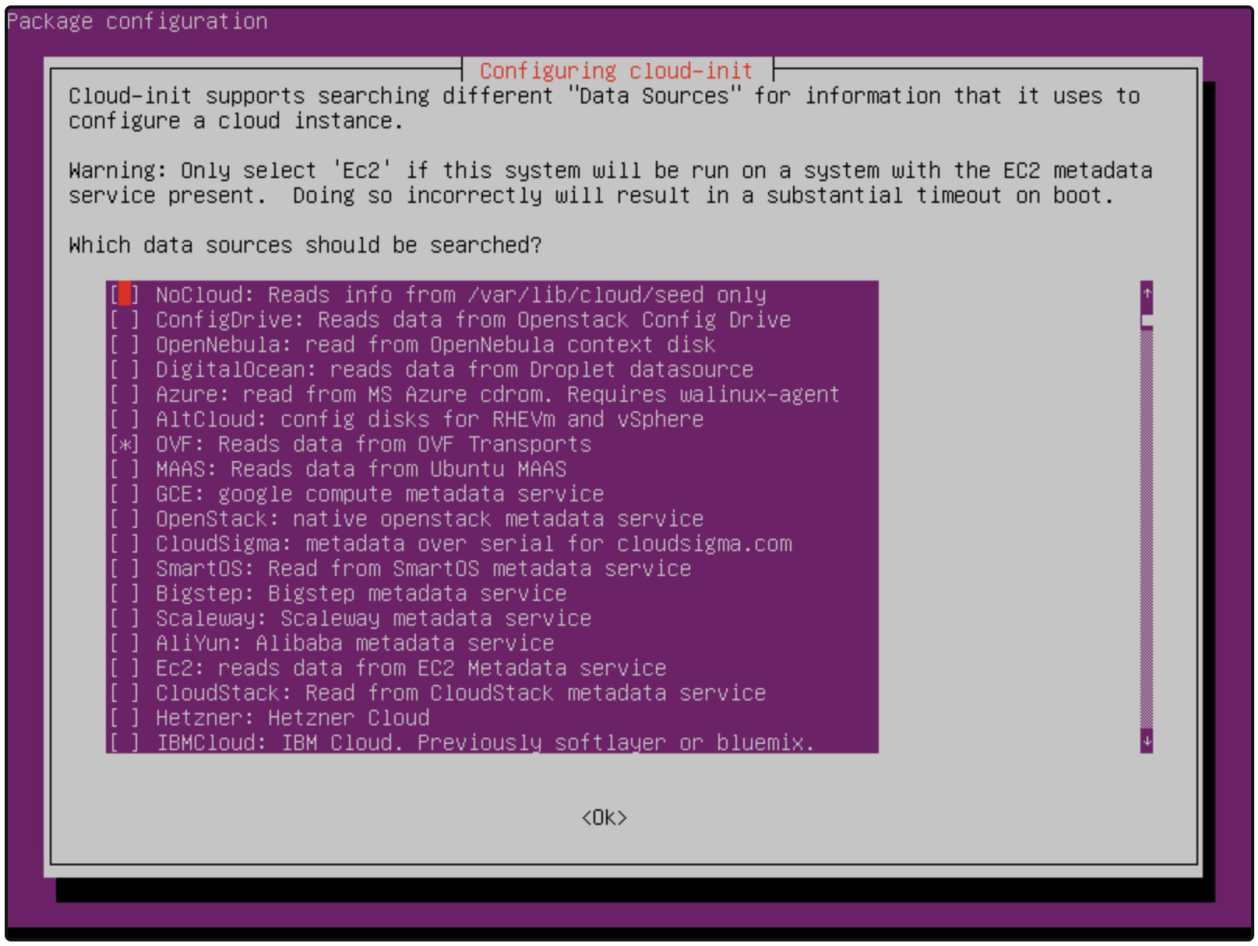
1. Start the virtual machine.
2. Wait for the IP address to be set.
3. Forward port `22` to the virtual machine:



Log on to the virtual machine over SSH and run the following commands:

```
rm /etc/netplan/99-netcfg-vmware.yaml
echo -e '\n[deployPkg]\nwait-cloudinit-timeout=1800\n' >> /etc/vmware-tools/tools.conf
echo 'disable_vmware_customization: true' > /etc/cloud/cloud.cfg.d/91_vmware_cust.cfg
dpkg-reconfigure cloud-init
```

In the dialog box that appears, leave the checkmark only on `OVF: Reads data from OVF transports`, and make sure to scroll down and remove checkmarks from other options:



Make sure that the `datasource_list` parameter is specified in the `cloud-init` configuration. You can verify this using the following command:

```
cat /etc/cloud/cloud.cfg.d/90_dpkg.cfg
```

If the output is empty, execute the following command:

```
echo "datasource_list: [ OVF, VMware, None ]" > /etc/cloud/cloud.cfg.d/90_dpkg.cfg
```

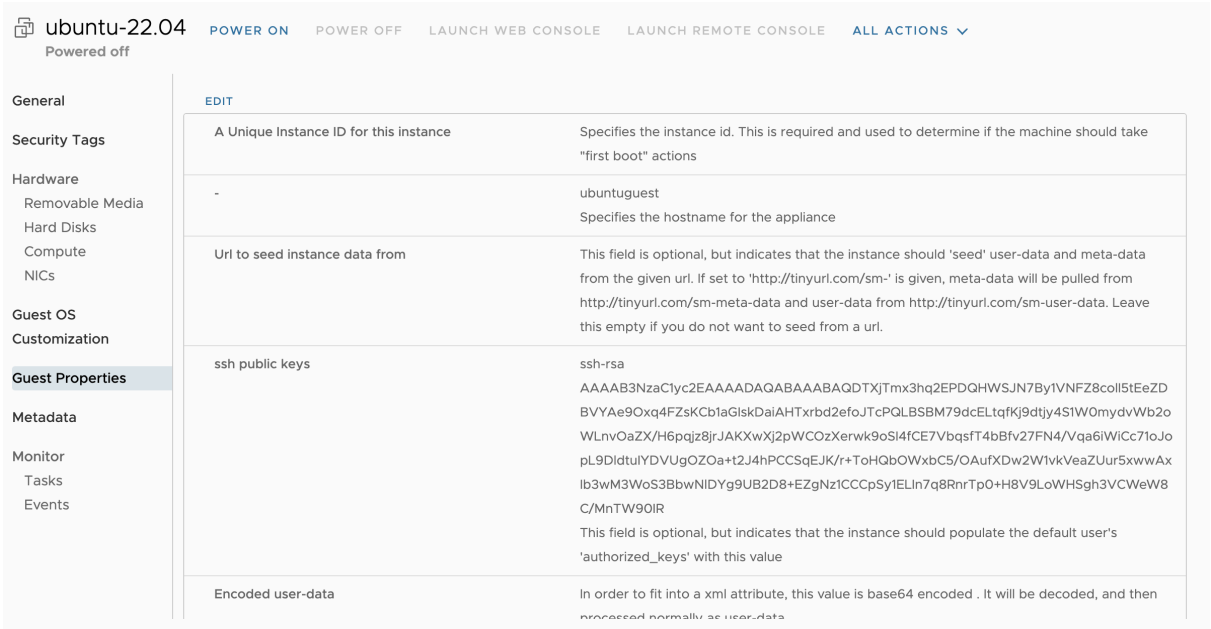
Execute the remaining commands:

```
truncate -s 0 /etc/machine-id
rm /var/lib/dbus/machine-id
ln -s /etc/machine-id /var/lib/dbus/machine-id
cloud-init clean --logs --seed
passwd -d ubuntu
passwd -d root
rm /home/ubuntu/.ssh/authorized_keys
history -c

shutdown -P now
```

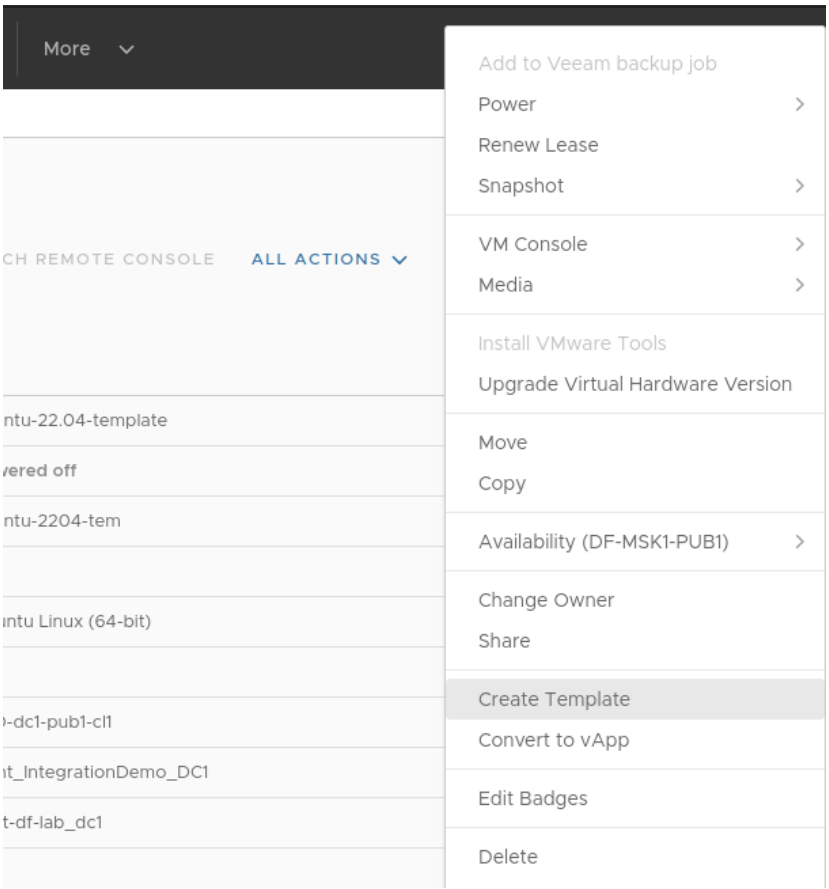
Setting up the template in VCD

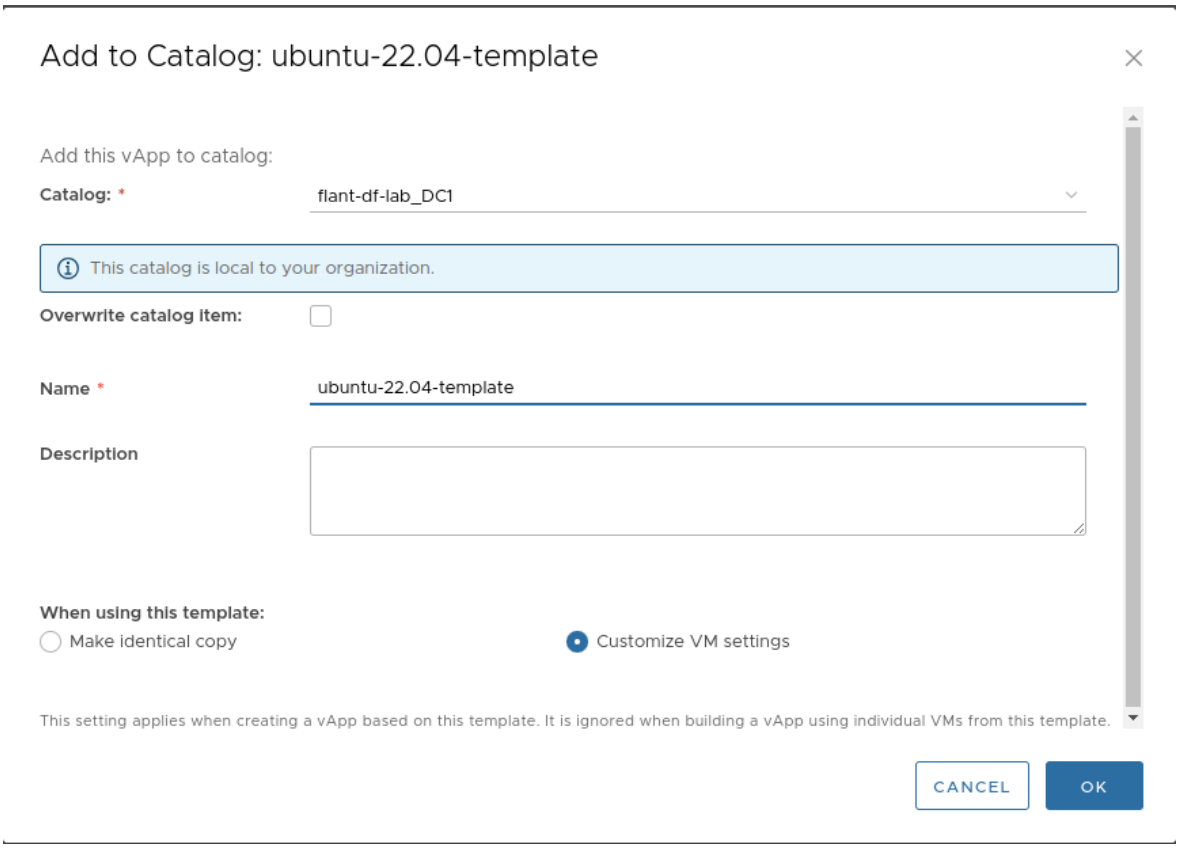
1. Shut down the virtual machine and clear all populated fields in “Guest Properties”:



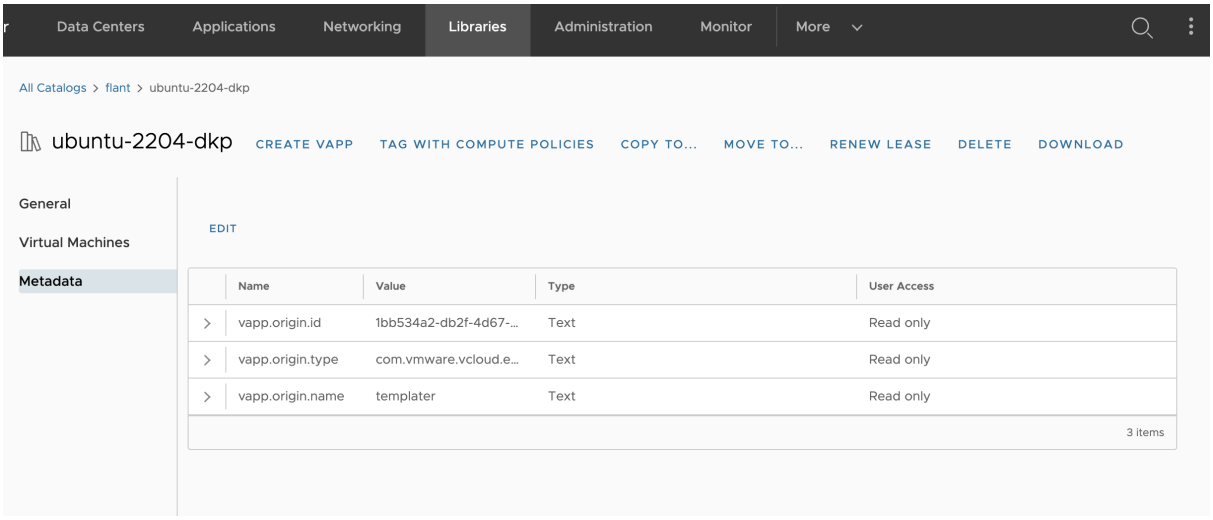


2. Create a virtual machine template:





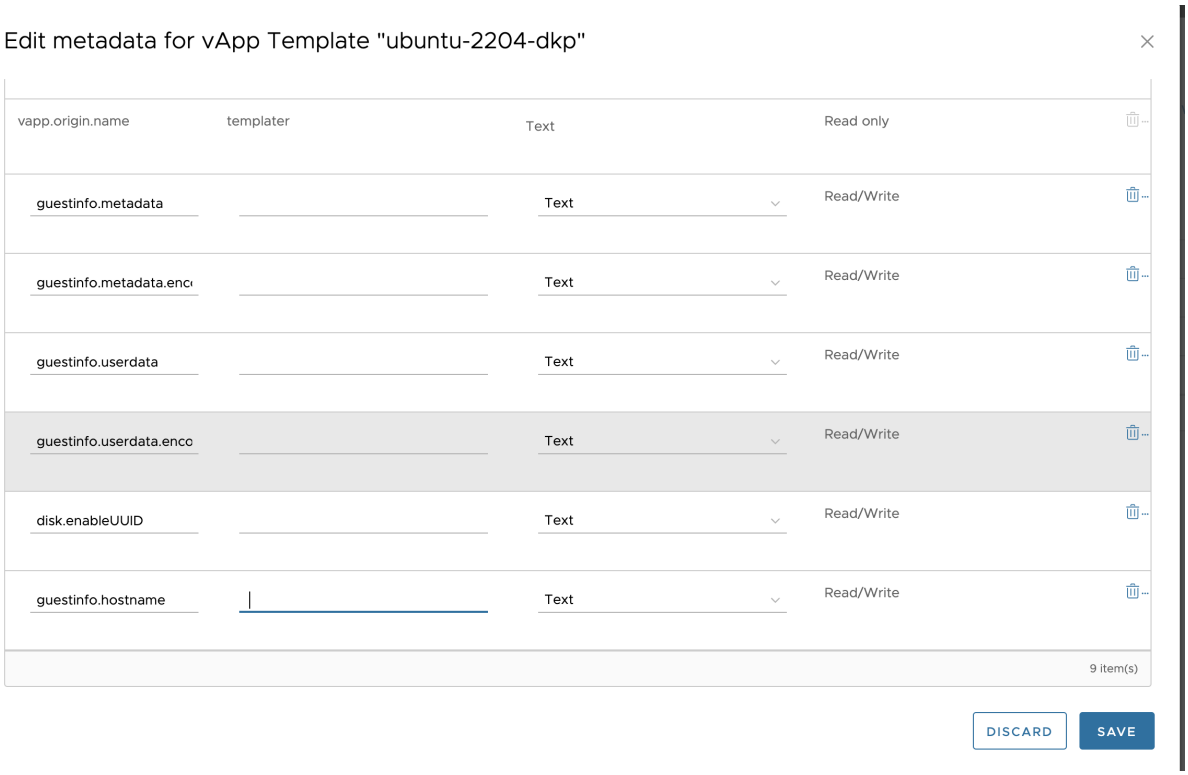
3. In the created template, navigate to the “Metadata” tab and add the following six fields:
   - `guestinfo.metadata`
   - `guestinfo.metadata.encoding`
   - `guestinfo.userdata`
   - `guestinfo.userdata.encoding`
   - `disk.enableUUID`
   - `guestinfo.hostname`



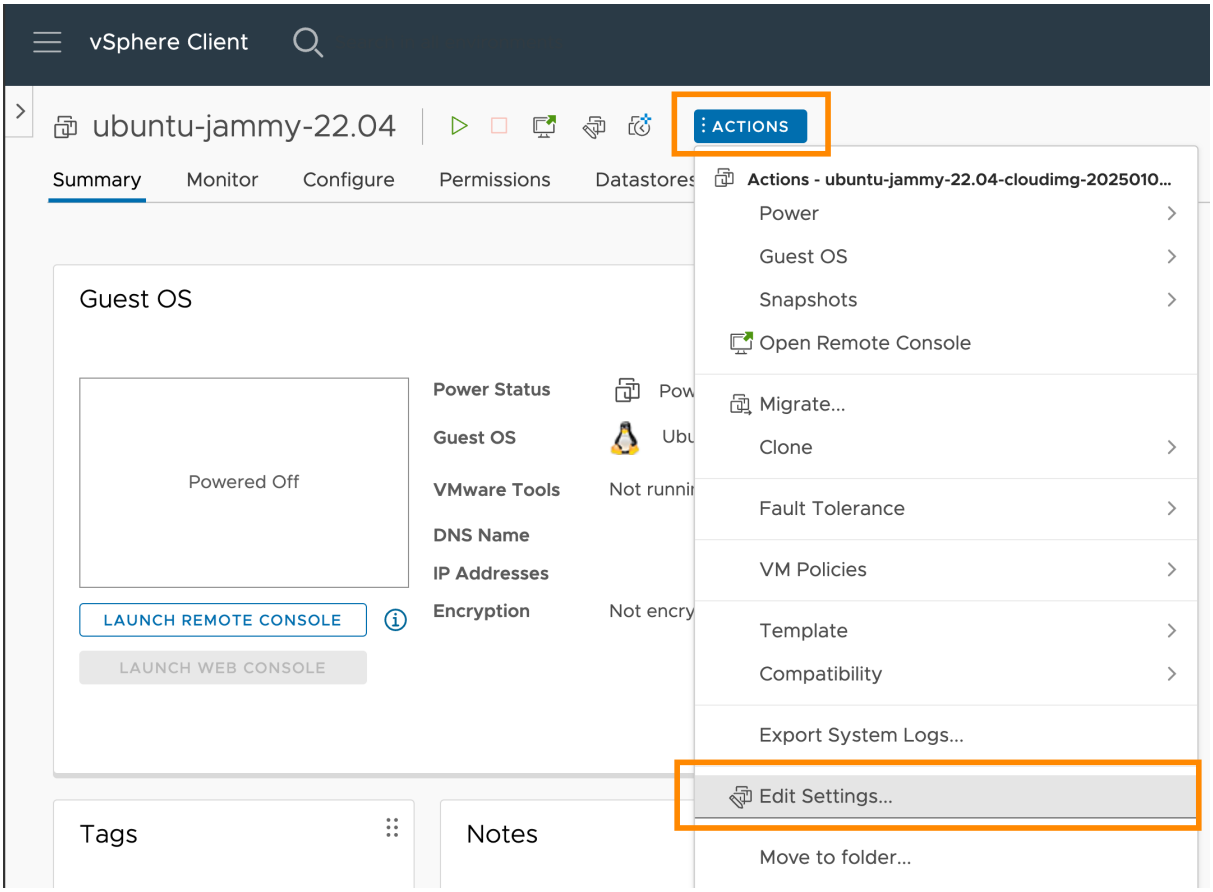
For **each** field in the add/edit form, set the following:

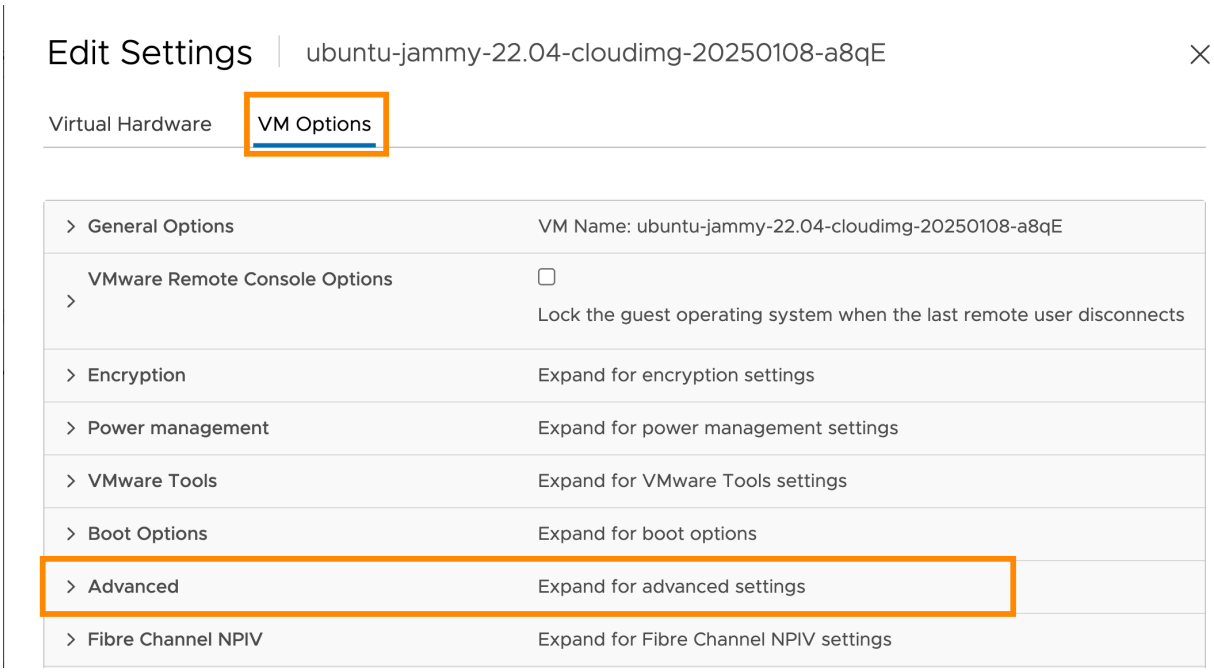
- “Type”: `Text` (text value)
- “User access”: `Read/Write`
- “Value”: A single space

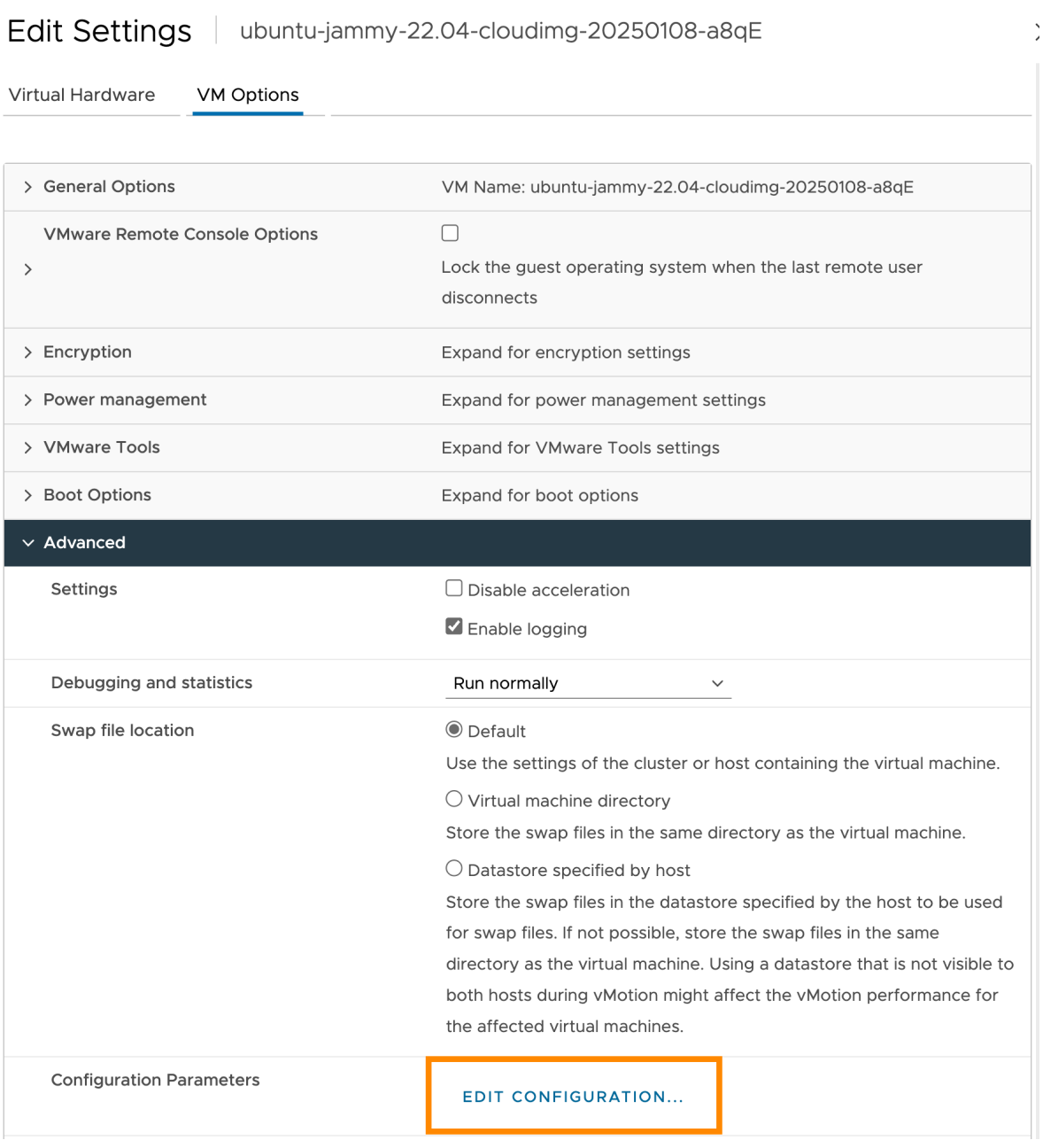
> **i** The VCD UI may not save metadata with an empty value. A single space is used as a placeholder and does not affect functionality. The actual values will be populated automatically when virtual machines are created.

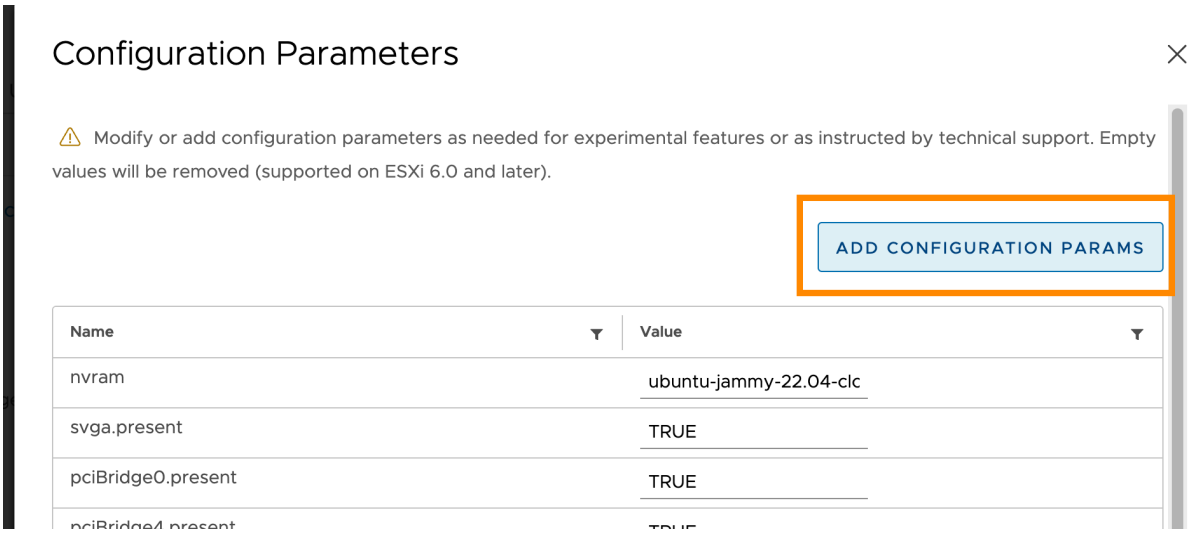


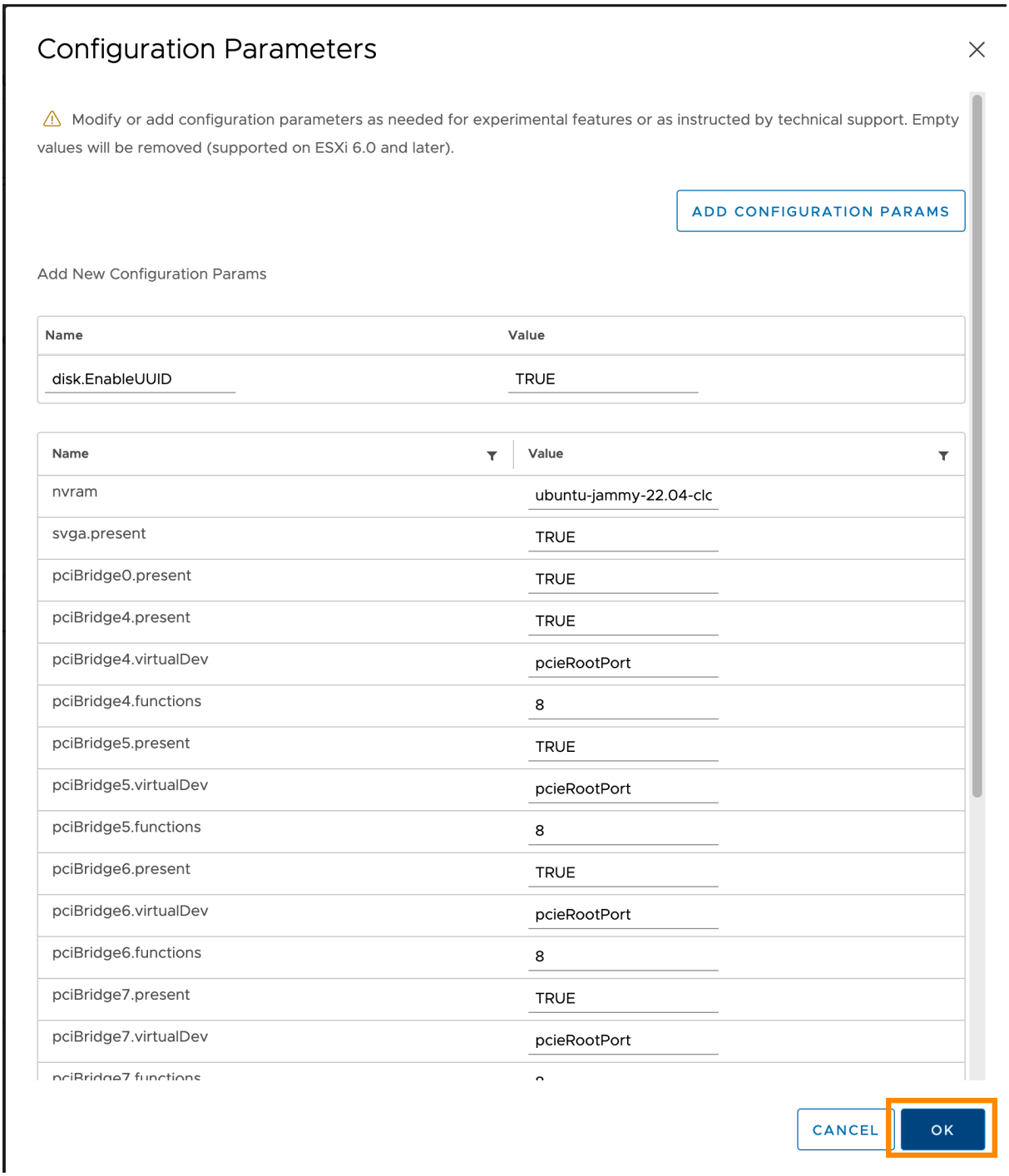
4. In the vCenter management panel for the template, enable the `disk.EnableUUID` parameter:




Edit Settings
ubuntu-jammy-22.04-cloudimg-20250108-a8qE
Virtual Hardware
VM Options
General Options
VM Name: ubuntu-jammy-22.04-cloudimg-20250108-a8qE
VMware Remote Console Options
Lock the guest operating system when the last remote user disconnects
Encryption
Expand for encryption settings
Power management
Expand for power management settings
VMware Tools
Expand for VMware Tools settings
Boot Options
Expand for boot options
Advanced
Expand for advanced settings
Fibre Channel NPIV
Expand for Fibre Channel NPIV settings


Edit Settings
ubuntu-jammy-22.04-cloudimg-20250108-a8qE
Virtual Hardware
VM Options
General Options
VM Name: ubuntu-jammy-22.04-cloudimg-20250108-a8qE
VMware Remote Console Options
Lock the guest operating system when the last remote user disconnects
Encryption
Expand for encryption settings
Power management
Expand for power management settings
VMware Tools
Expand for VMware Tools settings
Boot Options
Expand for boot options
Advanced
Settings
Disable acceleration
Enable logging
Debugging and statistics
Run normally
Swap file location
Default
Use the settings of the cluster or host containing the virtual machine.
Virtual machine directory
Store the swap files in the same directory as the virtual machine.
Datastore specified by host
Store the swap files in the datastore specified by the host to be used for swap files. If not possible, store the swap files in the same directory as the virtual machine. Using a datastore that is not visible to both hosts during vMotion might affect the vMotion performance for the affected virtual machines.
Configuration Parameters
EDIT CONFIGURATION...





Using the storage

- VCD supports CSI; disks are created as VCD Independent Disks.
- The `disk.EnableUUID` guest property must be set for the virtual machine templates in use.

- Deckhouse Kubernetes Platform supports disk resizing as of v1.59.1.

Using the LoadBalancer

- DKP components support `Service` resources of type `LoadBalancer` when deployed on VMware Cloud Director (VCD).
- VMware NSX Advanced Load Balancer (ALB or Avi Networks) is used as the load balancer.
- Support is available **only** when using the `NSX-T` network virtualization platform.
- The load balancer functionality must be enabled on the Edge Gateway by your VCD provider. You can verify this under **Edge Gateway → Load Balancer → General Settings** — the `State` parameter must be `Active`.
- If the load balancer was enabled after the DKP cluster was successfully created, the components will automatically pick up the changes within an hour (no additional actions are required).
- For each open port, a **Pool + Virtual Service** pair is created.
- If a firewall is in place, you must create an allow rule for the load balancer's external IP address and the corresponding ports.

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

VCDInstanceClass

Scope: Cluster
Version: v1

Parameters of a group of VCD VirtualMachines used by `cluster-api-controller-manager` (the cloud-provider-vcd module).

- **spec**
  object

  Required value

  - **spec.additionalMetadata**
    object

    Custom string key-value pairs to be stored in the Metadata of every node VM, useful for organizing and categorizing your virtualization inventory.

    The properties in this field will be merged with the existing metadata stored in the `metadata` field at the root of the VCDProviderClusterConfiguration spec. This field will take precedence over the existing metadata in cases of duplicate keys.

    > **i** **Warning.** Changing the `additionalMetadata` will cause the recreation of CloudEphemeral NodeGroups referring the VCDInstanceClass.

  - **spec.affinityRule**
    object

    Specifies the affinity rule applied only to virtual machines of node groups referencing this instance class within the cluster. Affinity rules determine how nodes are scheduled in relation to each other at the hypervisor level, which can help improve fault tolerance, and control the distribution of workloads. If not specified, no affinity rule will be applied by default.

    - **spec.affinityRule.polarity**
      string

      Required value

      The polarity of the affinity rule. Must be either `Affinity` or `AntiAffinity`. `Affinity` means that nodes should be placed on the same host, while `AntiAffinity` means they should be placed on different hosts.

      Allowed values: `Affinity`, `AntiAffinity`

    - **spec.affinityRule.required**
      boolean

      Indicates whether the affinity rule is mandatory (`true`) or only preferred (`false`). If set to `true`, the rule must be enforced; if `false`, the rule is applied if possible but not strictly required.

      Default: `false`

  - **spec.mainNetwork**
    string

    Path to the network that VirtualMachines' primary NICs will connect to (default gateway).

    Example:

```
mainNetwork: internal
```

- **spec.placementPolicy**
  string

  PlacementPolicy is the placement policy to be used on this machine. If no placement policy is specified, default placement policy will be used to create the nodes.

  Example:

  ```
  placementPolicy: Hypervisor1
  ```

- **spec.rootDiskSizeGb**
  integer

  root disk size in gigabytes.

  Default: `20`

  Example:

  ```
  rootDiskSizeGb: 20
  ```

- **spec.sizingPolicy**
  string

  SizingPolicy is the sizing policy to be used on this machine. If no sizing policy is specified, default sizing policy will be used to create the nodes.

  Example:

  ```
  sizingPolicy: 4Cpu8Memory
  ```

- **spec.storageProfile**
  string

  Required value

  StorageProfile is the storage profile to be used on this machine.

  Example:

  ```
  storageProfile: ssd
  ```

- **spec.template**
  string

  Required value

  Path of the template OVA that is to be used. Consists of organization (optional), catalog, template name or template identifier. If a shared catalog is used, the organization name must be specified (it may differ from the value of `provider.organization`).

  If you have templates with the same name in multiple catalogs with the same name, you should specify the identifier of the template.

  You can find the template identifier in the URL when you open the template in the vCD UI. For example, if the URL is `https://vcd.example.com/tenant/MyOrg/libraries/catalogs/1032506a-4555-43e1-8589-77b0c0522c75/catalog-vapp-templates/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7/general`, then the template identifier is `5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7`.

  Examples:

  ```
  template: Linux/ubuntu2204-cloud-ova
  ```

  ```
  template: MyOrg/Linux/ubuntu2204-cloud-ova
  ```

  ```
  template: Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
  ```

  ```
  template: MyOrg/Linux/5438c9f2-7e59-4eb3-b37c-cbd7f3e710c7
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Examples

> **i** The module lifecycle stage: General Availability

Below is an example VCDInstanceClass configuration for ephemeral nodes of VMware Cloud Director cloud provider.

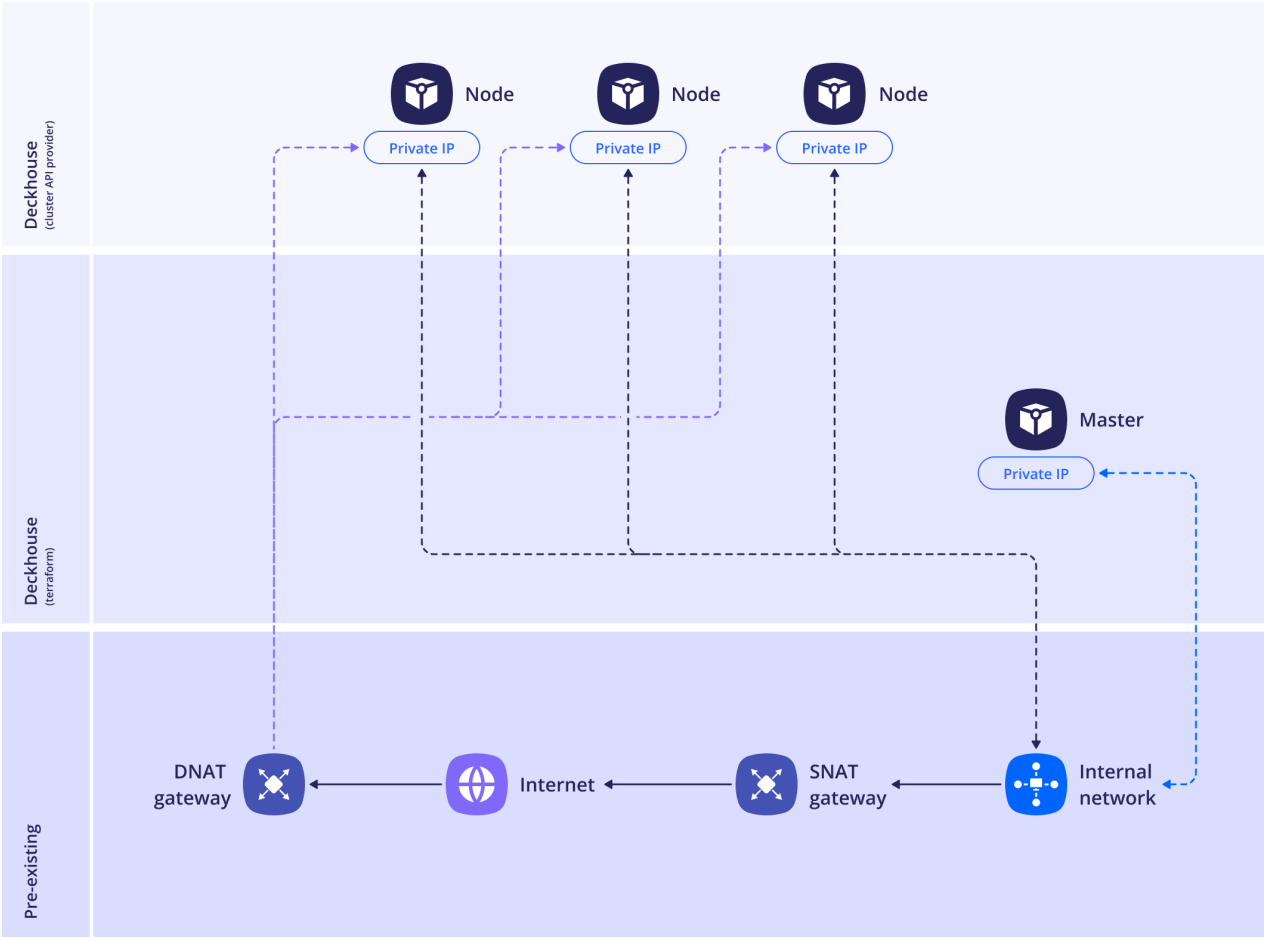
Example configuration of the VCDInstanceClass custom resource

```
apiVersion: deckhouse.io/v1
kind: VCDInstanceClass
metadata:
  name: test
spec:
  rootDiskSizeGb: 90
  sizingPolicy: payg-4-8
  storageProfile: SSD-dc1-pub1-cl1
  template: MyOrg/Linux/ubuntu2204-cloud-ova
```

Cloud provider - VMware Cloud Director: Layouts

> **i** The module lifecycle stage: General Availability

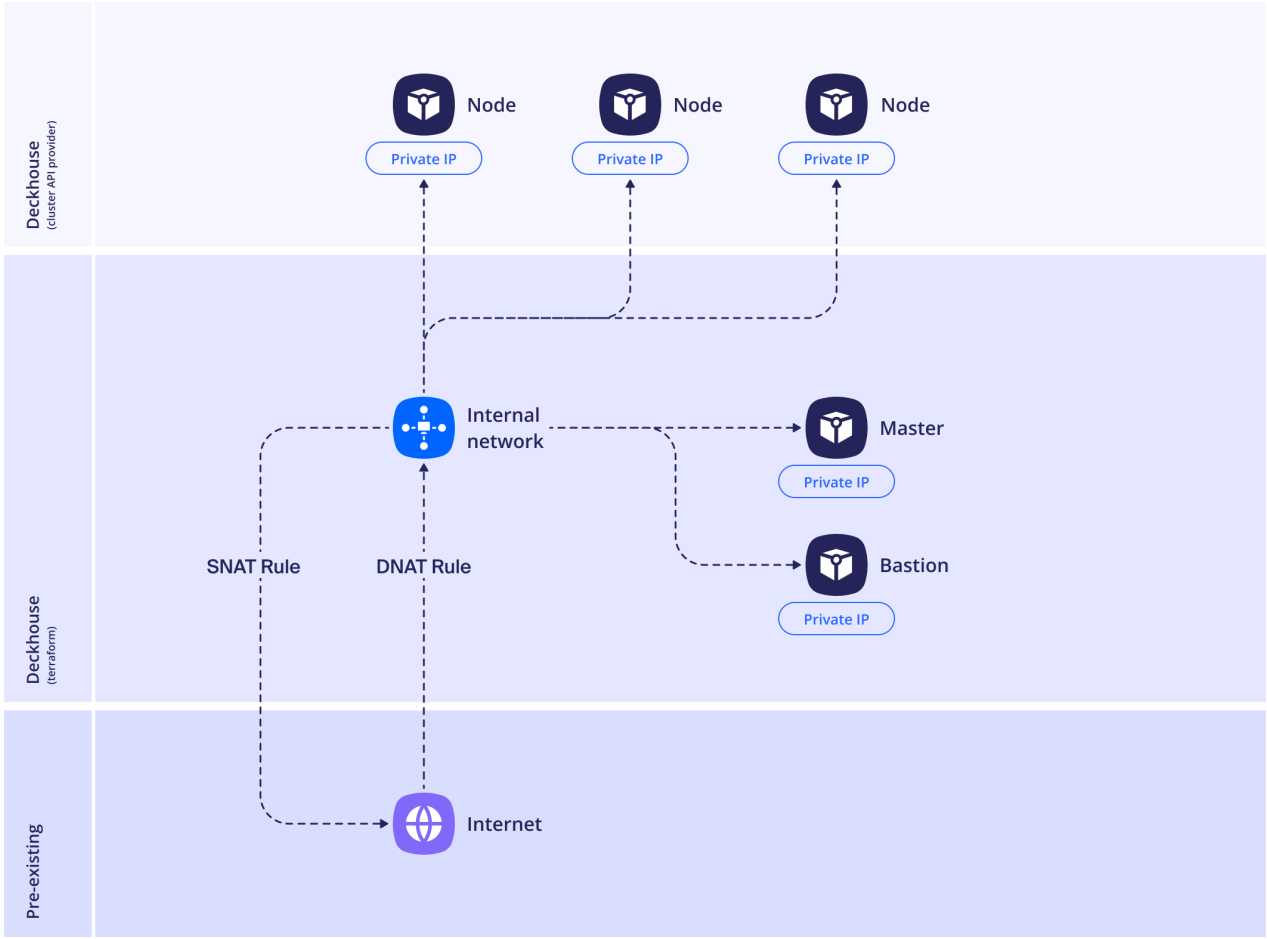
Standard



Example of the layout configuration:

```
---
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: Standard
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
    - 192.168.199.2
```

WithNAT



When using this placement scheme, you must check with the administrator which network virtualization platform is in use and specify it in the `edgeGateway.type` parameter.
Two options are supported: `NSX-T` and `NSX-V`.

To ensure administrative access to the cluster nodes, a bastion is deployed. The parameters for its configuration are described in the `bastion` section.

If the Edge Gateway is based on `NSX-T`, a DHCP server will be automatically enabled in the created network for the nodes. It will assign IP addresses starting from the 30th address in the subnet up to the second-to-last (just before the broadcast address).
You can change the starting address of the DHCP pool using the `internalNetworkDHCPPoolStartAddress` parameter.

If `NSX-V` is used, DHCP must be configured manually. Otherwise, nodes that rely on dynamic IP assignment will not be able to obtain an address.

⚠ It is not recommended to use dynamic addressing for the first master node in combination with `NSX-V`.

The deployment scheme assumes automated creation of NAT rules:

- An SNAT rule for translating the addresses of the internal node network to the external address specified in the `edgeGateway.externalIP` property.
- A DNAT rule for translating the external address and port, specified in the `edgeGateway.externalIP` and `edgeGateway.externalPort` properties, respectively, to the internal address of the bastion instance on port 22 using the `TCP` protocol for administrative access to the nodes via SSH.

⚠ If the Edge Gateway is provided by `NSX-V`, you must specify the name and type of the network to which the rule will be bound in the `edgeGateway.NSX-V.externalNetworkName` and `edgeGateway.NSX-V.externalNetworkType` properties, respectively. Typically, this is a network connected to the Edge Gateway in `Gateway Interface` and having an external IP address.

Additionally, you can enable the creation of default firewall rules using the `createDefaultFirewallRules` property.

⚠ If the Edge Gateway is provided by `NSX-T`, existing rules on the Edge Gateway will be overwritten. It is assumed that using this option implies that only one cluster will be deployed per Edge Gateway.

The following rules will be created:

- Allow any outgoing traffic
- Allow incoming traffic over the `TCP` protocol on port 22 to enable SSH access to the cluster nodes
- Allow any incoming traffic over the `ICMP` protocol
- Allow incoming traffic over the `TCP` and `UDP` protocols on ports 30000–32767 for NodePort usage

Example of the layout configuration using `NSX-T`:

```
---
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: WithNAT
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
internalNetworkDNSServers:
  - 77.88.8.8
  - 1.1.1.1
mainNetwork: internal
bastion:
  instanceClass:
    rootDiskSizeGb: 30
    sizingPolicy: 2cpu1mem
    template: "catalog/Ubuntu 22.04 Server"
    storageProfile: Fast vHDD
    mainNetworkIPAddress: 10.1.4.10
edgeGateway:
  name: "edge-gateway-01"
  type: "NSX-T"
  externalIP: 10.0.0.1
  externalPort: 10022
createDefaultFirewallRules: false
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
    - 192.168.199.2
```

Example of the layout configuration using `NSX-V`:

```
---
apiVersion: deckhouse.io/v1alpha1
kind: VCDClusterConfiguration
layout: WithNAT
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
sshPublicKey: <SSH_PUBLIC_KEY>
organization: deckhouse
virtualDataCenter: MSK-1
virtualApplicationName: deckhouse
internalNetworkCIDR: 192.168.199.0/24
internalNetworkDNSServers:
  - 77.88.8.8
  - 1.1.1.1
mainNetwork: internal
bastion:
  instanceClass:
    rootDiskSizeGb: 30
    sizingPolicy: 2cpu1mem
    template: "catalog/Ubuntu 22.04 Server"
    storageProfile: Fast vHDD
    mainNetworkIPAddress: 10.1.4.10
edgeGateway:
  name: "edge-gateway-01"
  type: "NSX-V"
  externalIP: 10.0.0.1
  externalPort: 10022
  NSX-V:
    externalNetworkName: external
    externalNetworkType: ext
createDefaultFirewallRules: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    storageProfile: "Fast vHDD"
    sizingPolicy: 4cpu8mem
    template: "catalog/Ubuntu 22.04 Server"
    mainNetworkIPAddresses:
    - 192.168.199.2
```

Cloud provider — VMware vSphere

> **i** The module lifecycle stage: General Availability

The `cloud-provider-vsphere` module integrates Deckhouse Kubernetes Platform with VMware vSphere. It allows the `node-manager` module to use vSphere resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-vsphere` module:

- Managing vSphere resources via `cloud-controller-manager`:
  - creates network routes for the `PodNetwork` network on the vSphere side;
  - updates virtual machine and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in vSphere.
- Provisioning disks via CSI on datastore. By default, CNS volumes with online resize are used. First-Class Disk (FCD) mode is available as legacy and is configured with the `compatibilityFlag` parameter.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-vsphere`.
- Provisioning CloudEphemeral nodes via Machine Controller Manager (MCM). Virtual machine parameters are set in the VsphereInstanceClass resource.
- Registering with `node-manager` so that VsphereInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

Configuration

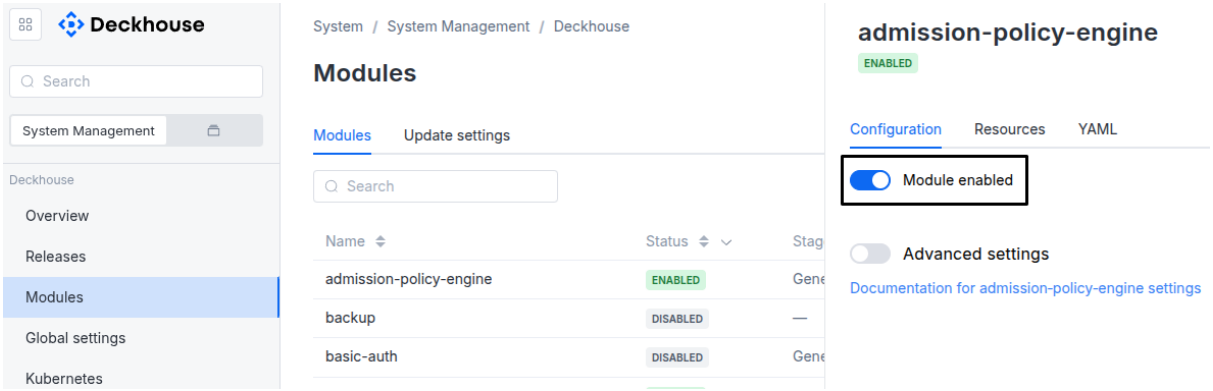
> **i** The module lifecycle stage: General Availability

The module is automatically enabled for all cloud clusters deployed in vSphere.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-vsphere` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cloud-provider-vsphere
```

- **Using ModuleConfig `cloud-provider-vsphere`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-vsphere` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-vsphere`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  enabled: true
```

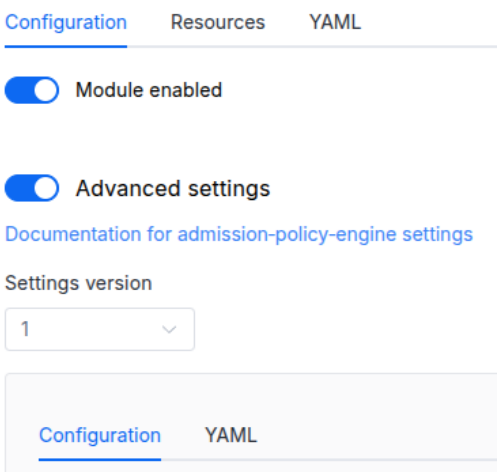
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-vsphere` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `cloud-provider-vsphere` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-vsphere`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-vsphere` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc cloud-provider-vsphere
```

Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `cloud-provider-vsphere` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `cloud-provider-vsphere`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  version: 2
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  If the field `.storageClass.compatibilityFlag` exists, capitalize the field value.

If the cluster control plane is hosted on a virtual machines or bare-metal servers, the cloud provider uses the settings from the `cloud-provider-vsphere` module in the Deckhouse configuration (see below). Otherwise, if the cluster control plane is hosted in a cloud, the cloud provider uses the VsphereClusterConfiguration structure for configuration.

You can configure the number and parameters of ordering machines in the cloud via the `NodeGroup` custom resource of the `node-manager` module. Also, in this custom resource, you can specify the instance class's name for the above group of nodes (the `cloudInstances.ClassReference` parameter of NodeGroup). In the case of the vSphere cloud provider, the instance class is the `VsphereInstanceClass` custom resource that stores specific parameters of the machines.

Storage

The module automatically creates a StorageClass for each Datastore and DatastoreCluster in the zone (or zones).

Also, it can set the name of StorageClass that will be used in the cluster by default (the default parameter), and filter out the unnecessary StorageClasses (the exclude parameter).

CSI

By default, the storage subsystem uses CNS volumes with the ability of online-resize. FCD volumes are also supported, but only in the legacy or migration modes. You can set this via the compatibilityFlag parameter.

Important information concerning the increase of the PVC size

Due to the nature f volume-resizer, CSI, and vSphere API, you have to do the following after increasing the PVC size:

1. On the node where the Pod is located, run the `d8 k cordon <node_name>` command.
2. Delete the Pod.
3. Make sure that the resize was successful. The PVC object must *not have* the `Resizing` condition.

   > **i** The `FileSystemResizePending` state is OK.

4. On the node where the Pod is located, run the `d8 k uncordon <node_name>` command.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.disableTimesync**
    boolean

    Disable time synchronization on the vSphere side.

> **Caution!** This parameter will not disable the NTP daemons in the guest OS, but only disable the time correction on the part of ESXi.

- **settings.externalNetworkNames**
  array of strings

  Names of networks (just the name and not the full path) connected to `VirtualMachines` and used by `vsphere-cloud-controller-manager` to insert ExternalIP into the `.status.addresses` field in the Node API object.

- **settings.host**
  string

  The domain of the vCenter server.

- **settings.insecure**
  boolean

  Set to `true` if vCenter has a self-signed certificate.

- **settings.internalNetworkNames**
  array of strings

  Names of networks (just the name and not the full path) connected to `VirtualMachines` and used by `vsphere-cloud-controller-manager` to insert InternalIP into the `.status.addresses` field in the Node API object.

- **settings.nsxt**
  object

  Kubernetes load balancer support using NSX-T for the vSphere cloud controller manager.

  - **settings.nsxt.defaultIpPoolName**
    string

    Required value

    Name of the default IP pool used for the SVC's without `loadbalancer.vmware.io/class` annotation set.

    Example:

    ```
    defaultIpPoolName: pool1
    ```

  - **settings.nsxt.defaultTcpAppProfileName**
    string

    Name of default NSX-T application profile used for TCP connections.

    Default: `default-tcp-lb-app-profile`

    Examples:

    ```
    defaultTcpAppProfileName: default-tcp-lb-app-profile
    ```

    ```
    defaultTcpAppProfileName: tcp-profile1
    ```

  - **settings.nsxt.defaultUdpAppProfileName**
    string

    Name of default NSX-T application profile used for UDP connections.

    Default: `default-udp-lb-app-profile`

    Examples:

    ```
    defaultUdpAppProfileName: default-udp-lb-app-profile
    ```

    ```
    defaultUdpAppProfileName: udp-profile1
    ```

  - **settings.nsxt.host**
    string

    Required value

    NSX-T host.

    Example:

    ```
    host: 1.2.3.4
    ```

  - **settings.nsxt.insecureFlag**
    boolean

    To be set to true if NSX-T uses self-signed certificate.

    Examples:

```
insecureFlag: true
```

```
insecureFlag: false
```

- **settings.nsxt.loadBalancerClass**
  array

  Additional section to define Load Balancer Classes (to use class, set annotation `loadbalancer.vmware.io/class: <class name>` to SVC).

  Examples:

```
loadBalancerClass: []
```

```
loadBalancerClass:
  name: LBC1
  ipPoolName: pool2
```

```
loadBalancerClass:
  name: LBC1
  ipPoolName: pool2
  tcpAppProfileName: profile2
  udpAppProfileName: profile3
```

  - **settings.nsxt.loadBalancerClass.ipPoolName**
    string

    Required value

    Name of the IP pool.

  - **settings.nsxt.loadBalancerClass.name**
    string

    Required value

    Load Balancer Class name to use in SVC annotation `loadbalancer.vmware.io/class: <class name>`.

  - **settings.nsxt.loadBalancerClass.tcpAppProfileName**
    string

    Name of application profile used for TCP connections.

    Default: `defaultTcpAppProfileName`

  - **settings.nsxt.loadBalancerClass.udpAppProfileName**
    string

    Name of application profile used for UDP connections.

    Default: `defaultUdpAppProfileName`

- **settings.nsxt.password**
  string

  Required value

  NSX-T password.

  Example:

```
password: password
```

- **settings.nsxt.size**
  string

  Size of load balancer service.

  Default: `MEDIUM`

  Allowed values: `SMALL`, `MEDIUM`, `LARGE`, `XLARGE`

  Example:

```
size: SMALL
```

- **settings.nsxt.tier1GatewayPath**
  string

  Required value

  Policy path for the NSX-T tier1 gateway.

Example:

```
tier1GatewayPath: "/path/tier1"
```

- **settings.nsxt.user**
  string

  Required value

  NSX-T user name.

  Example:

  ```
  user: user
  ```

- **settings.password**
  string

  The user's password.

- **settings.region**
  string

  Is a tag added to the vSphere Datacenter where all actions will occur: provisioning VirtualMachines, storing virtual disks on datastores, connecting to the network.

- **settings.regionTagCategory**
  string

  The name of the tag category used to identify the region (vSphere Datacenter).

- **settings.sshKeys**
  array of strings

  A list of public SSH keys in plain-text format.

- **settings.storageClass**
  object
  - **settings.storageClass.compatibilityFlag**
    string

    A flag allowing the use of the old CSI version:

    - `Legacy` — use the old version of the driver. FCD discs only, no online-resizing;
    - `Migration` — in this case, both drivers will be available in the cluster at the same time. This mode is used to migrate from an old driver.

    Allowed values: `Legacy`, `Migration`

  - **settings.storageClass.default**
    Deprecated
    string

    The name of StorageClass that will be used in the cluster by default.

    If the parameter is omitted, the default StorageClass is either:

    - an arbitrary StorageClass present in the cluster that has the default annotation;
    - the first (in lexicographic order) StorageClass created by the module.

    > **i** **Parameter is deprecated.** Instead, use the global parameter global.defaultClusterStorageClass.

    Example:

    ```
    default: fast-lun102-7d0bf578
    ```

  - **settings.storageClass.exclude**
    array of strings

    A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

    Example:

    ```
    exclude:
    - ".*-lun101-.*"
    - slow-lun103-1c280603
    ```

- **settings.username**
  string

  Username.

> **Warning.** Make sure to specify the username together with the domain, for example: `username@domain.local`.
>
> You can find the correct domain in the vCenter control panel.

- **settings.vmFolderPath**
  string

  The path to the VirtualMachine Folder where the cloned VMs will be created.
- **settings.zoneTagCategory**
  string

  The name of the tag category used to identify the zone (vSphere Cluster).
- **settings.zones**
  array of strings

  The globally restricted set of zones that this Cloud Provider works with.

Provider configuration

> The module lifecycle stage: General Availability

> If the cluster control plane is hosted on a virtual machines or bare-metal servers, the cloud provider uses the settings from the `cloud-provider-vsphere` module in the Deckhouse configuration. Otherwise, if the cluster control plane is hosted in a cloud, the cloud provider uses the VsphereClusterConfiguration structure for configuration.
>
> Additional info about Vsphere Cloud Load Balancers.

VsphereClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in vSphere.

Used by the cloud provider if a cluster’s control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

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

> After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes section. Information about reducing the number of nodes and deleting nodes is provided in the Integration with VMware vSphere services section.

Example:

```
apiVersion: deckhouse.io/v1
kind: VsphereClusterConfiguration
sshPublicKey: "<SSH_PUBLIC_KEY>"
layout: Standard
vmFolderPath: folder/prefix
regionTagCategory: k8s-region
zoneTagCategory: k8s-zone
region: region2
zones:
- region2-a
externalNetworkNames:
- net3-k8s
internalNetworkNames:
- K8S_3
internalNetworkCIDR: 172.16.2.0/24
baseResourcePool: kubernetes/cloud
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: Templates/ubuntu-focal-20.04
    mainNetwork: net3-k8s
    additionalNetworks:
    - K8S_3
    datastore: lun10
    rootDiskSize: 50
    runtimeOptions:
      nestedHardwareVirtualization: false
nodeGroups:
- name: worker
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: Templates/ubuntu-focal-20.04
    datastore: lun10
    mainNetwork: net3-k8s
provider:
  server: "<SERVER>"
  username: "<USERNAME@DOMAIN.LOCAL>"
  password: "<PASSWORD>"
  insecure: true
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **baseResourcePool**
  string

  A path (relative to vSphere Cluster) to the existing parent `resourcePool` for all `resourcePool` created in each zone.

- **disableTimesync**
  boolean

  Disable time synchronization on the vSphere side.

  > **Caution!** This parameter will not disable the NTP daemons in the guest OS, but only disable the time correction on the part of ESXi.

  Default: `true`

- **externalNetworkNames**
  array of strings

  Names of networks (just the name and not the full path) connected to `VirtualMachines` and used by `vsphere-cloud-controller-manager` to insert ExternalIP into the `.status.addresses` field in the Node API object.

  Example:

```
externalNetworkNames:
- MAIN-1
- public
```

- **internalNetworkCIDR**
  string

  Subnet for master nodes in the internal network.

  Addresses are allocated starting with the tenth address. E.g., if you have the `192.168.199.0/24` subnet, addresses will be allocated starting with `192.168.199.10`.

  The `internalNetworkCIDR` is used if `additionalNetworks` are defined in `masterInstanceClass`.

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **internalNetworkNames**
  array of strings

  Names of networks (just the name and not the full path) connected to `VirtualMachines` and used by `vsphere-cloud-controller-manager` to insert InternalIP into the `.status.addresses` field in the Node API object.

  Example:

```
internalNetworkNames:
- KUBE-3
- devops-internal
```

- **kind**
  string

  Required value

  Allowed values: `VsphereClusterConfiguration`

- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

- **masterNodeGroup**
  object

  Required value

  The definition of the master's NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the fields of the VsphereInstanceClass.

    - **masterNodeGroup.instanceClass.additionalNetworks**
      array of strings

      Paths to networks that VirtualMachines' secondary NICs will connect to. Relative to the datacenter.

      Example:

```
additionalNetworks:
- DEVOPS_32
- DEVOPS_50
```

    - **masterNodeGroup.instanceClass.datastore**
      string

      Required value

      Path to a Datastore in which VirtualMachines will be cloned. Relative to the datacenter.

      Example:

```
datastore: lun-1201
```

    - **masterNodeGroup.instanceClass.mainNetwork**

string

Required value

Path to the network that VirtualMachines' primary NICs will connect to (default gateway). Relative to the datacenter.

Example:

```
mainNetwork: k8s-msk-178
```

- **masterNodeGroup.instanceClass.mainNetworkIPAddresses**
  array of objects

  A list of static IP addresses (with a CIDR prefix) sequentially allocated to nodes in the `mainNetwork`.

  By default, the DHCP client is enabled.

  Example:

  ```
  mainNetworkIPAddresses:
  - address: 10.1.14.20/24
    gateway: 10.1.14.254
    nameservers:
      addresses:
      - 8.8.8.8
      - 8.8.4.4
  ```

  - **masterNodeGroup.instanceClass.mainNetworkIPAddresses.address**
    string

    Required value

    An IP address with a CIDR prefix.

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  - **masterNodeGroup.instanceClass.mainNetworkIPAddresses.gateway**
    string

    Required value

    The IP address of the default gateway.

    It must be located in the subnet specified in the `address` parameter

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

  - **masterNodeGroup.instanceClass.mainNetworkIPAddresses.nameservers**
    object

    Required value

    - **masterNodeGroup.instanceClass.mainNetworkIPAddresses.nameservers.addresses**
      array of strings

      A list of DNS servers.

      Example:

      ```
      addresses:
      - 8.8.8.8
      - 8.8.4.4
      ```

      - **Element of the array**
        string

        Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

    - **masterNodeGroup.instanceClass.mainNetworkIPAddresses.nameservers.search**
      array of strings

      A list of DNS search domains.

      Example:

      ```
      search:
      - tech.lan
      ```

- **masterNodeGroup.instanceClass.memory**
  integer

  Required value

Memory in MiB to allocate to vSphere VirtualMachines.

Example:

```
memory: 8192
```

- **masterNodeGroup.instanceClass.numCPUs**
integer

  Required value

  Count of vCPUs to allocate to vSphere VirtualMachines.

  Example:

  ```
  numCPUs: 2
  ```

- **masterNodeGroup.instanceClass.resourcePool**
string

  Path to a Resource Pool in which VirtualMachines will be cloned. Relative to the zone (vSphere Cluster).

  Example:

  ```
  resourcePool: rp-2012
  ```

- **masterNodeGroup.instanceClass.rootDiskSize**
integer

  Root disk size in GiB to use in vSphere VirtualMachines.

  The disk will be automatically enlarged if its size in the template is less than specified.

  Example:

  ```
  rootDiskSize: 50
  ```

- **masterNodeGroup.instanceClass.runtimeOptions**
object

  Additional VM’s parameters.

  - **masterNodeGroup.instanceClass.runtimeOptions.cpuLimit**
  integer

    CPU limit in MHz.

  - **masterNodeGroup.instanceClass.runtimeOptions.cpuReservation**
  integer

    CPU reservation in MHz.

  - **masterNodeGroup.instanceClass.runtimeOptions.cpuShares**
  integer

    The relative amount of CPU Shares for VMs to be created.

  - **masterNodeGroup.instanceClass.runtimeOptions.memoryLimit**
  integer

    Memory limit in MB.

  - **masterNodeGroup.instanceClass.runtimeOptions.memoryReservation**
  integer

    VM memory reservation in percent (relative to `.spec.memory`).

    Allowed values: `0 <= X <= 100`

  - **masterNodeGroup.instanceClass.runtimeOptions.memoryShares**
  integer

    The relative amount of Memory Shares for VMs to be created.

    Allowed values: `0 <= X <= 100`

  - **masterNodeGroup.instanceClass.runtimeOptions.nestedHardwareVirtualization**
  boolean

    Whether to enable or disable nested Hardware Assisted Virtualization.

- **masterNodeGroup.instanceClass.template**
string

  Required value

  Path to the template to be cloned. Relative to the datacenter.

  Example:

```
template: dev/golden_image
```

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.zones**
  array of strings

  A limited set of zones in which master nodes can be created.

  The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the VsphereInstanceClass.

    - **nodeGroups.instanceClass.additionalNetworks**
      array of strings

      Paths to networks that VirtualMachines' secondary NICs will connect to. Relative to the datacenter.

      Example:

      ```
      additionalNetworks:
      - DEVOPS_32
      - DEVOPS_50
      ```

    - **nodeGroups.instanceClass.datastore**
      string

      Required value

      Path to a Datastore in which VirtualMachines will be cloned. Relative to the datacenter.

      Example:

      ```
      datastore: lun-1201
      ```

    - **nodeGroups.instanceClass.mainNetwork**
      string

      Required value

      Path to the network that VirtualMachines' primary NICs will connect to (default gateway). Relative to the datacenter.

      Example:

      ```
      mainNetwork: k8s-msk-178
      ```

    - **nodeGroups.instanceClass.mainNetworkIPAddresses**
      array of objects

      A list of static IP addresses (with a CIDR prefix) sequentially allocated to nodes in the `mainNetwork`.

      By default, the DHCP client is enabled.

      Example:

      ```
      mainNetworkIPAddresses:
      - address: 10.1.14.20/24
        gateway: 10.1.14.254
        nameservers:
          addresses:
          - 8.8.8.8
          - 8.8.4.4
      ```

      - **nodeGroups.instanceClass.mainNetworkIPAddresses.address**
        string

Required value

An IP address with a CIDR prefix.

Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **nodeGroups.instanceClass.mainNetworkIPAddresses.gateway**
  string

  Required value

  The IP address of the default gateway.

  It must be located in the subnet specified in the `address` parameter

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

- **nodeGroups.instanceClass.mainNetworkIPAddresses.nameservers**
  object

  Required value

  - **nodeGroups.instanceClass.mainNetworkIPAddresses.nameservers.addresses**
    array of strings

    A list of DNS servers.

    Example:

    ```
    addresses:
    - 8.8.8.8
    - 8.8.4.4
    ```

    - **Element of the array**
      string

      Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$`

  - **nodeGroups.instanceClass.mainNetworkIPAddresses.nameservers.search**
    array of strings

    A list of DNS search domains.

    Example:

    ```
    search:
    - tech.lan
    ```

- **nodeGroups.instanceClass.memory**
  integer

  Required value

  Memory in MiB to allocate to vSphere VirtualMachines.

  Example:

  ```
  memory: 8192
  ```

- **nodeGroups.instanceClass.numCPUs**
  integer

  Required value

  Count of vCPUs to allocate to vSphere VirtualMachines.

  Example:

  ```
  numCPUs: 2
  ```

- **nodeGroups.instanceClass.resourcePool**
  string

  Path to a Resource Pool in which VirtualMachines will be cloned. Relative to the zone (vSphere Cluster).

  Example:

  ```
  resourcePool: rp-2012
  ```

- **nodeGroups.instanceClass.rootDiskSize**
  integer

  Root disk size in GiB to use in vSphere VirtualMachines.

The disk will be automatically enlarged if its size in the template is less than specified.

Example:

```
rootDiskSize: 50
```

- **nodeGroups.instanceClass.runtimeOptions**
  object

  Additional VM’s parameters.

  - **nodeGroups.instanceClass.runtimeOptions.cpuLimit**
    integer

    CPU limit in MHz.

  - **nodeGroups.instanceClass.runtimeOptions.cpuReservation**
    integer

    CPU reservation in MHz.

  - **nodeGroups.instanceClass.runtimeOptions.cpuShares**
    integer

    The relative amount of CPU Shares for VMs to be created.

  - **nodeGroups.instanceClass.runtimeOptions.memoryLimit**
    integer

    Memory limit in MB.

  - **nodeGroups.instanceClass.runtimeOptions.memoryReservation**
    integer

    VM memory reservation in percent (relative to `.spec.memory`).

    Allowed values: `0 <= X <= 100`

  - **nodeGroups.instanceClass.runtimeOptions.memoryShares**
    integer

    The relative amount of Memory Shares for VMs to be created.

    Allowed values: `0 <= X <= 100`

  - **nodeGroups.instanceClass.runtimeOptions.nestedHardwareVirtualization**
    boolean

    Whether to enable or disable nested Hardware Assisted Virtualization.

- **nodeGroups.instanceClass.template**
  string

  Required value

  Path to the template to be cloned. Relative to the datacenter.

  Example:

  ```
  template: dev/golden_image
  ```

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: 'true'
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to cluster resources.

    The same as the `metadata.labels` standard field.

    Example:

```
labels:
  environment: production
  app: warp-drive-ai
```

- **nodeGroups.nodeTemplate.taints**
  array of objects

  The same as the `.spec.taints` field of the Node object.

  > **i** Available fields: `effect`, `key`, and `values`.

  Example:

  ```
  taints:
  - effect: NoExecute
    key: ship-class
    value: frigate
  ```

  - **nodeGroups.nodeTemplate.taints.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
  - **nodeGroups.nodeTemplate.taints.key**
    string
  - **nodeGroups.nodeTemplate.taints.value**
    string

- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes to create.

- **nodeGroups.zones**
  array of strings

  A limited set of zones in which nodes can be created.

- **nsxt**
  object

  Kubernetes load balancer support using NSX-T for the vSphere cloud controller manager.

  - **nsxt.defaultIpPoolName**
    string

    Required value

    Name of the default IP pool used for the SVC's without `loadbalancer.vmware.io/class` annotation set.

    Example:

    ```
    defaultIpPoolName: pool1
    ```

  - **nsxt.defaultTcpAppProfileName**
    string

    Name of default NSX-T application profile used for TCP connections.

    Default: `default-tcp-lb-app-profile`

    Examples:

    ```
    defaultTcpAppProfileName: default-tcp-lb-app-profile
    ```

    ```
    defaultTcpAppProfileName: tcp-profile1
    ```

  - **nsxt.defaultUdpAppProfileName**
    string

    Name of default NSX-T application profile used for UDP connections.

    Default: `default-udp-lb-app-profile`

    Examples:

    ```
    defaultUdpAppProfileName: default-udp-lb-app-profile
    ```

    ```
    defaultUdpAppProfileName: udp-profile1
    ```

  - **nsxt.host**

string

Required value

NSX-T host.

Example:

```
host: 1.2.3.4
```

- **nsxt.insecureFlag**
  boolean

  To be set to true if NSX-T uses self-signed certificate.

  Examples:

  ```
  insecureFlag: true
  ```

  ```
  insecureFlag: false
  ```

- **nsxt.loadBalancerClass**
  array

  Additional section to define Load Balancer Classes (set annotation `loadbalancer.vmware.io/class: <CLASS NAME>` to SVC to use the class).

  Examples:

  ```
  loadBalancerClass: []
  ```

  ```
  loadBalancerClass:
    name: LBC1
    ipPoolName: pool2
  ```

  ```
  loadBalancerClass:
    name: LBC1
    ipPoolName: pool2
    tcpAppProfileName: profile2
    udpAppProfileName: profile3
  ```

  - **nsxt.loadBalancerClass.ipPoolName**
    string

    Required value

    Name of the IP pool.

  - **nsxt.loadBalancerClass.name**
    string

    Required value

    Load Balancer Class name to use in SVC annotation `loadbalancer.vmware.io/class: <CLASS NAME>`.

  - **nsxt.loadBalancerClass.tcpAppProfileName**
    string

    Name of application profile used for TCP connections.

    Default: `defaultTcpAppProfileName`

  - **nsxt.loadBalancerClass.udpAppProfileName**
    string

    Name of application profile used for UDP connections.

    Default: `defaultUdpAppProfileName`

- **nsxt.password**
  string

  Required value

  NSX-T password.

  Example:

  ```
  password: password
  ```

- **nsxt.size**
  string

  Size of load balancer service.

Default: `MEDIUM`

Allowed values: `SMALL`, `MEDIUM`, `LARGE`, `XLARGE`

Example:

```
size: SMALL
```

- **nsxt.tier1GatewayPath**
  string

  Required value

  Policy path for the NSX-T tier1 gateway.

  Example:

  ```
  tier1GatewayPath: "/path/tier1"
  ```

- **nsxt.user**
  string

  Required value

  NSX-T user name.

  Example:

  ```
  user: user
  ```

- **provider**
  object

  Required value

  Parameters for connecting to the vCenter.

  - **provider.insecure**
    boolean

    Set to `true` if vCenter has a self-signed certificate.

    Default: `false`

  - **provider.password**
    string

    Required value

    The user’s password.

  - **provider.server**
    string

    Required value

    The host or the IP address of the vCenter server.

  - **provider.username**
    string

    Required value

    Username.

    > **Warning.** Make sure to specify the username together with the domain, for example: `username@domain.local`.
    >
    > You can find the correct domain in the vCenter control panel.

- **region**
  string

  Required value

  Is a tag added to the vSphere Datacenter where all actions will occur: provisioning VirtualMachines, storing virtual disks on datastores, connecting to the network.

- **regionTagCategory**
  string

  Required value

  The name of the tag category used to identify the region (vSphere Datacenter).

  Default: `k8s-region`

- **sshPublicKey**

string

Required value

A public key for accessing nodes.

- **storagePolicyID**
  string

  ID of the SPBM (Storage Policy Based Management) storage policy.

- **useNestedResourcePool**
  boolean

  Create nested resource pool (`true`) or use main resource pool (`false`).

  Default: `true`

- **vmFolderExists**
  boolean

  Set the value to `true` if the path specified in `vmFolderPath` exists. Installing more than one cluster in a folder is not possible.

  Default: `false`

- **vmFolderPath**
  string

  Required value

  The path to the VirtualMachine Folder where the cloned VMs will be created.

  Example:

  ```
  vmFolderPath: dev/test
  ```

- **zoneTagCategory**
  string

  Required value

  The name of the tag category used to identify the zone (vSphere Cluster).

  Default: `k8s-zone`

- **zones**
  array of strings

  Required value

  The globally restricted set of zones that this Cloud Provider works with.

Preparing environment

> **i** The module lifecycle stage: General Availability

Environment requirements

The following prerequisites must be met for Deckhouse Kubernetes Platform to work correctly with VMware vSphere:

- Access to vCenter;
- A user account with the required set of privileges;
- Tags and tag categories created in vSphere;
- Networks with DHCP and Internet access;
- Shared Datastore resources available on all ESXi hosts in use;
- vSphere version `7.x` or `8.x` with support for `Online volume expansion`;
- vCenter accessible from inside the cluster from the master nodes;
- A configured Datacenter that includes the following objects:
  1. VirtualMachine template:
     - The virtual machine image must use `Virtual machines with hardware version 15 or later` — this is required for online resize support.
     - The image must include the `open-vm-tools`, `cloud-init`, and `cloud-init-vmware-guestinfo` packages if the `cloud-init` version is lower than `21.3`.
  2. Network:
     - The network must be available on all ESXi hosts where virtual machines are planned to be created.
  3. Datastore (one or more):
     - The Datastore must be connected to all ESXi hosts where virtual machines are planned to be created.
     - A tag from the category specified in the `zoneTagCategory` parameter must be assigned to the Datastore (default: `k8s-zone`). This tag defines the zone.
     - All Cluster objects within the same zone must have access to all Datastore objects in that zone.
  4. Cluster:
     - All ESXi hosts in use must be added to the Cluster.
     - A tag from the category specified in the `zoneTagCategory` parameter must be assigned to the Cluster (default: `k8s-zone`). This tag defines the zone.
  5. Folder for the virtual machines being created:

    - This parameter is optional.
    - By default, the root virtual machine folder is used.
  6. Role:
    - The role must include the required set of privileges.
  7. User:
    - The user must be assigned the role specified in the previous item.
- A tag from the category specified in the `regionTagCategory` parameter must be assigned to the created Datacenter (default: `k8s-region`). This tag defines the region.

List of required vSphere resources

- **User** with required set of privileges.
- **Network** with DHCP server and access to the Internet.
- **Datacenter** with a tag in `k8s-region` category.
- **Cluster** with a tag in `k8s-zone` category.
- **Datastore** with required tags.
- **Template** — prepared VM image.

List of required privileges

> **i** Read the Configuration via vSphere Client and Configuration via govc sections for details on how to create and assign a role to a user.

A detailed list of privileges required for Deckhouse Kubernetes Platform to work in vSphere:

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

—	— (assigned by default when creating a role)	`System.Anonymous`
		`System.Read`
		`System.View`
			Basic access to vSphere Inventory objects required for all Deckhouse vSphere integration components.

Cns	Searchable	`Cns.Searchable`	Search and mapping of Container Native Storage objects when the CSI driver works with Kubernetes volumes.

Datastore	Allocate space,
	Browse datastore,
	Low level file operations
		`Datastore.AllocateSpace`
		`Datastore.Browse`
		`Datastore.FileManagement`
			Disk provisioning when creating virtual machines and ordering `PersistentVolumes` in the cluster.

Folder	Create folder,
	Delete folder,
	Move folder,
	Rename folder
		`Folder.Create`
		`Folder.Delete`
		`Folder.Move`
		`Folder.Rename`
			Grouping a Deckhouse Kubernetes Platform cluster in a single `Folder` in vSphere Inventory.

Global	Global tag, System tag	`Global.GlobalTag` `Global.SystemTag`	Access to global and system tags used by Deckhouse Kubernetes Platform when working with vSphere objects.

vSphere Tagging	Assign or Unassign vSphere Tag,
	Assign or Unassign vSphere Tag on Object,
	Create vSphere Tag,
	Create vSphere Tag Category,
	Delete vSphere Tag,
	Delete vSphere Tag Category,
	Edit vSphere Tag,
	Edit vSphere Tag Category,
	Modify UsedBy Field for Category,
	Modify UsedBy Field for Tag
		`InventoryService.Tagging.AttachTag`
		`InventoryService.Tagging.ObjectAttachable`
		`InventoryService.Tagging.CreateTag`
		`InventoryService.Tagging.CreateCategory`
		`InventoryService.Tagging.DeleteTag`
		`InventoryService.Tagging.DeleteCategory`
		`InventoryService.Tagging.EditTag`
		`InventoryService.Tagging.EditCategory`
		`InventoryService.Tagging.ModifyUsedByForCategory`
		`InventoryService.Tagging.ModifyUsedByForTag`
			Deckhouse Kubernetes Platform uses tags to identify the `Datacenter`, `Cluster`, and `Datastore` objects available to it, as well as to identify the virtual machines under its control.

Network	Assign network	`Network.Assign`	Connecting networks and port groups to Deckhouse Kubernetes Platform cluster virtual machines.

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

Resource	Assign virtual machine to resource pool,
	Create resource pool,
	Modify resource pool,
	Remove resource pool,
	Rename resource pool
		`Resource.AssignVMToPool`
		`Resource.CreatePool`
		`Resource.DeletePool`
		`Resource.EditPool`
		`Resource.RenamePool`
			Placement of Deckhouse Kubernetes Platform cluster virtual machines into the target resource pool and management of this pool.

VM Storage Policies (*Profile-driven Storage Privileges* in vSphere 7)	View VM storage policies (*Profile-driven storage view* in vSphere 7)	`StorageProfile.View`	Viewing storage policies used when creating virtual machines and dynamically provisioning volumes in the cluster.

vApp	Add virtual machine,
	Assign resource pool,
	Create,
	Delete,
	Import,
	Power Off,
	Power On,
	View OVF Environment,
	vApp application configuration,
	vApp instance configuration,
	vApp resource configuration
		`VApp.ApplicationConfig`
		`VApp.AssignResourcePool`
		`VApp.AssignVM`
		`VApp.Create`
		`VApp.Delete`
		`VApp.ExtractOvfEnvironment`
		`VApp.Import`
		`VApp.InstanceConfig`
		`VApp.PowerOff`
		`VApp.PowerOn`
		`VApp.ResourceConfig`
			Managing operations related to deployment and configuration of vApp and OVF templates used when creating virtual machines.

Virtual Machine > Change Configuration	Add existing disk,
	Add new disk,
	Add or remove device,
	Advanced configuration,
	Set annotation,
	Change CPU count,
	Toggle disk change tracking,
	Extend virtual disk,
	Acquire disk lease,
	Modify device settings,
	Configure managedBy,
	Change Memory,
	Query unowned files,
	Configure Raw device,
	Reload from path,
	Remove disk,
	Rename,
	Reset guest information,
	Change resource,
	Change Settings,
	Change Swapfile placement,
	Upgrade virtual machine compatibility
		`VirtualMachine.Config.AddExistingDisk`
		`VirtualMachine.Config.AddNewDisk`
		`VirtualMachine.Config.AddRemoveDevice`
		`VirtualMachine.Config.AdvancedConfig`
		`VirtualMachine.Config.Annotation`
		`VirtualMachine.Config.CPUCount`
		`VirtualMachine.Config.ChangeTracking`
		`VirtualMachine.Config.DiskExtend`
		`VirtualMachine.Config.DiskLease`
		`VirtualMachine.Config.EditDevice`
		`VirtualMachine.Config.ManagedBy`
		`VirtualMachine.Config.Memory`
		`VirtualMachine.Config.QueryUnownedFiles`
		`VirtualMachine.Config.RawDevice`
		`VirtualMachine.Config.ReloadFromPath`
		`VirtualMachine.Config.RemoveDisk`
		`VirtualMachine.Config.Rename`
		`VirtualMachine.Config.ResetGuestInfo`
		`VirtualMachine.Config.Resource`
		`VirtualMachine.Config.Settings`
		`VirtualMachine.Config.SwapPlacement`
		`VirtualMachine.Config.UpgradeVirtualHardware`
			Managing the lifecycle of Deckhouse Kubernetes Platform cluster virtual machines.

Virtual Machine > Edit Inventory	Create new,
	Create from existing,
	Remove,
	Move
		`VirtualMachine.Inventory.Create`
		`VirtualMachine.Inventory.CreateFromExisting`
		`VirtualMachine.Inventory.Delete`
		`VirtualMachine.Inventory.Move`
			Creating, deleting, and moving Deckhouse Kubernetes Platform cluster virtual machines in vSphere Inventory.

Virtual Machine > Guest Operations	Guest Operation Queries	`VirtualMachine.GuestOperations.Query`	Retrieving information from the guest operating system of virtual machines.

Privilege category in UI	Privileges in UI	Privileges in API	Purpose in Deckhouse

Virtual Machine > Interaction	Answer question,
	Device connection,
	Guest operating system management by VIX API,
	Power Off,
	Power On,
	Reset,
	Configure CD media,
	Install VMware Tools
		`VirtualMachine.Interact.AnswerQuestion`
		`VirtualMachine.Interact.DeviceConnection`
		`VirtualMachine.Interact.GuestControl`
		`VirtualMachine.Interact.PowerOff`
		`VirtualMachine.Interact.PowerOn`
		`VirtualMachine.Interact.Reset`
		`VirtualMachine.Interact.SetCDMedia`
		`VirtualMachine.Interact.ToolsInstall`
			Managing virtual machine power state, device connections, and interaction with the guest operating system.

Virtual Machine > Provisioning	Clone virtual machine,
	Customize guest,
	Deploy template,
	Allow virtual machine download,
	Allow virtual machine files upload,
	Read customization specifications
		`VirtualMachine.Provisioning.Clone`
		`VirtualMachine.Provisioning.Customize`
		`VirtualMachine.Provisioning.DeployTemplate`
		`VirtualMachine.Provisioning.GetVmFiles`
		`VirtualMachine.Provisioning.PutVmFiles`
		`VirtualMachine.Provisioning.ReadCustSpecs`
			Cloning virtual machine templates, customizing them, and deploying them when creating Deckhouse Kubernetes Platform cluster nodes.

Virtual Machine > Snapshot Management	Create snapshot,
	Remove Snapshot,
	Rename Snapshot
		`VirtualMachine.State.CreateSnapshot`
		`VirtualMachine.State.RemoveSnapshot`
		`VirtualMachine.State.RenameSnapshot`
			Managing snapshots of virtual machines and volumes in scenarios where this functionality is used by platform components.

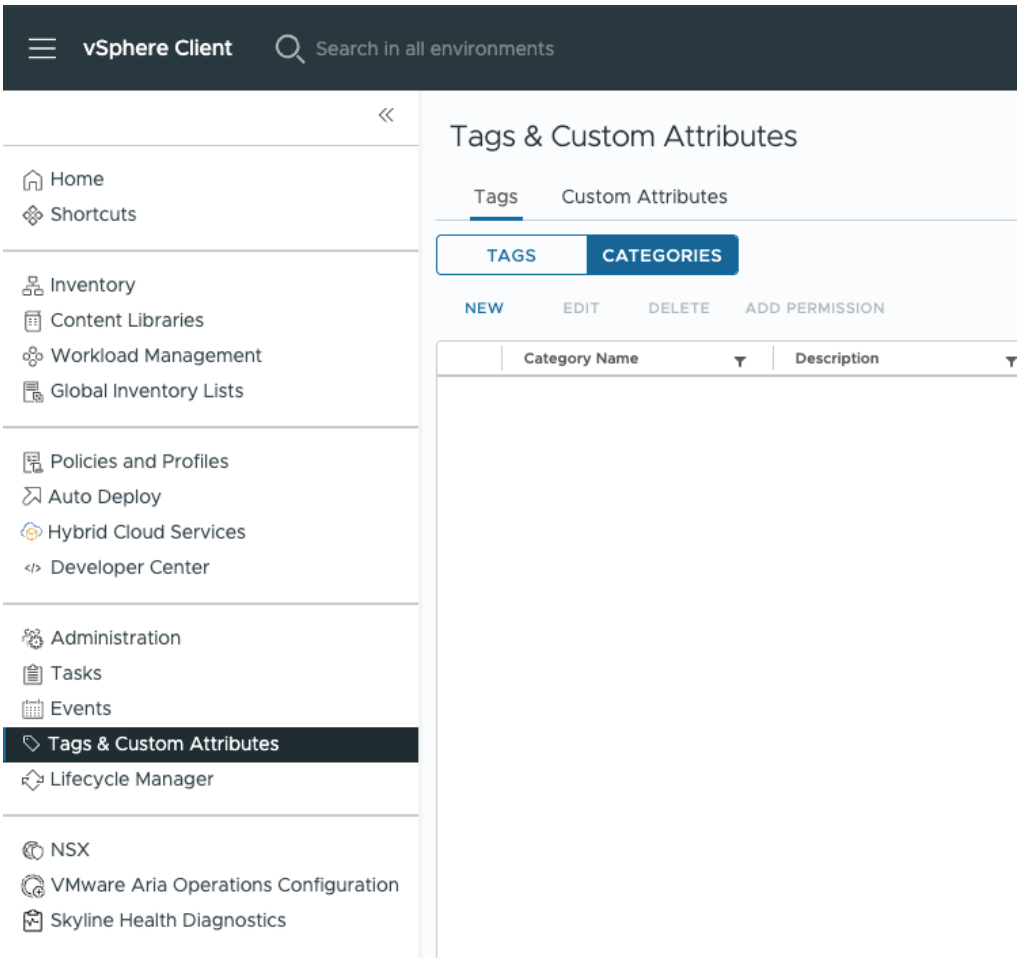
vSphere configuration

Configuration in vSphere Client

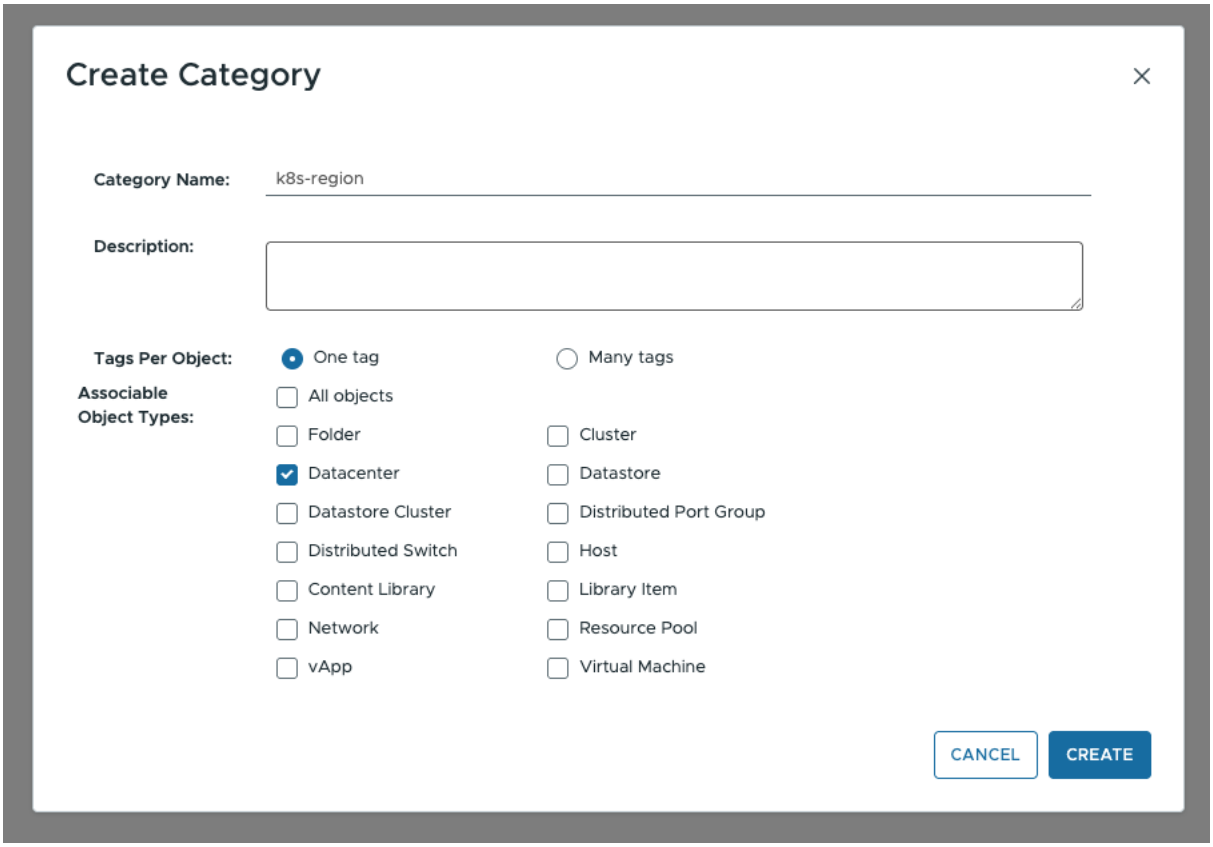
Creating tags and tag categories in vSphere Client

VMware vSphere does not have built-in concepts of a “region” or a “zone”. In vSphere, a Datacenter acts as a region, and a Cluster acts as a zone. Tags are used to establish this mapping.

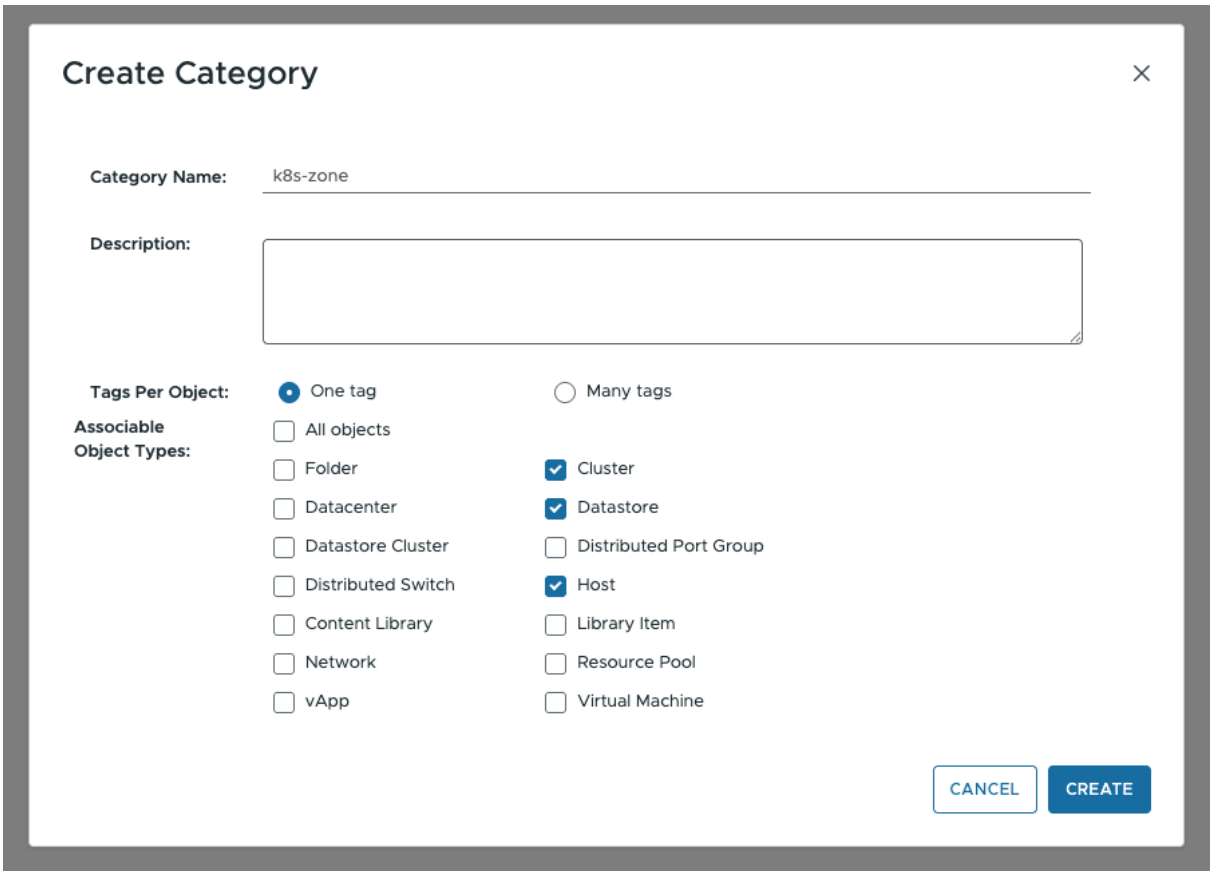
1. Open vSphere Client and go to “Menu” → “Tags & Custom Attributes” → “Tags”.



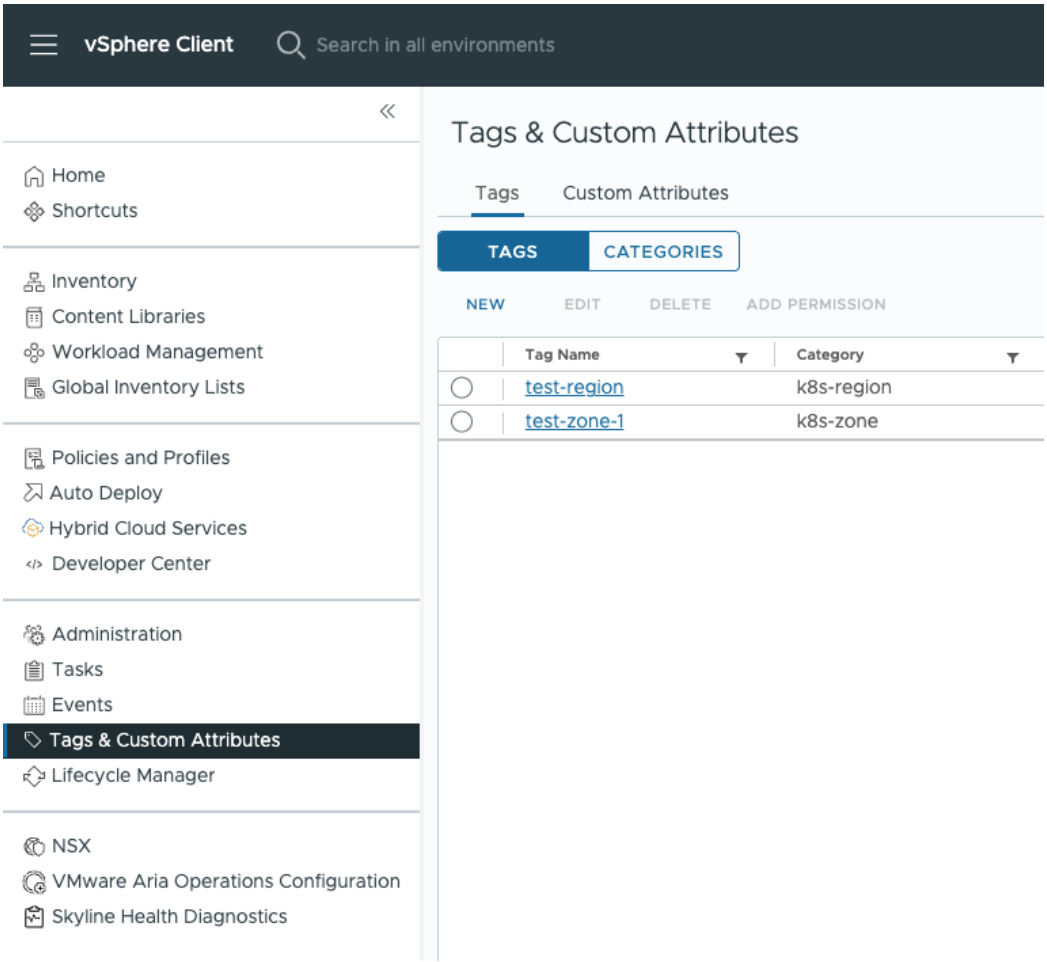
2. Open the “Categories” tab and click “NEW”. Create a category for regions (for example, `k8s-region`): set “Tags Per Object” to “One tag” and specify the applicable object types, including Datacenter.



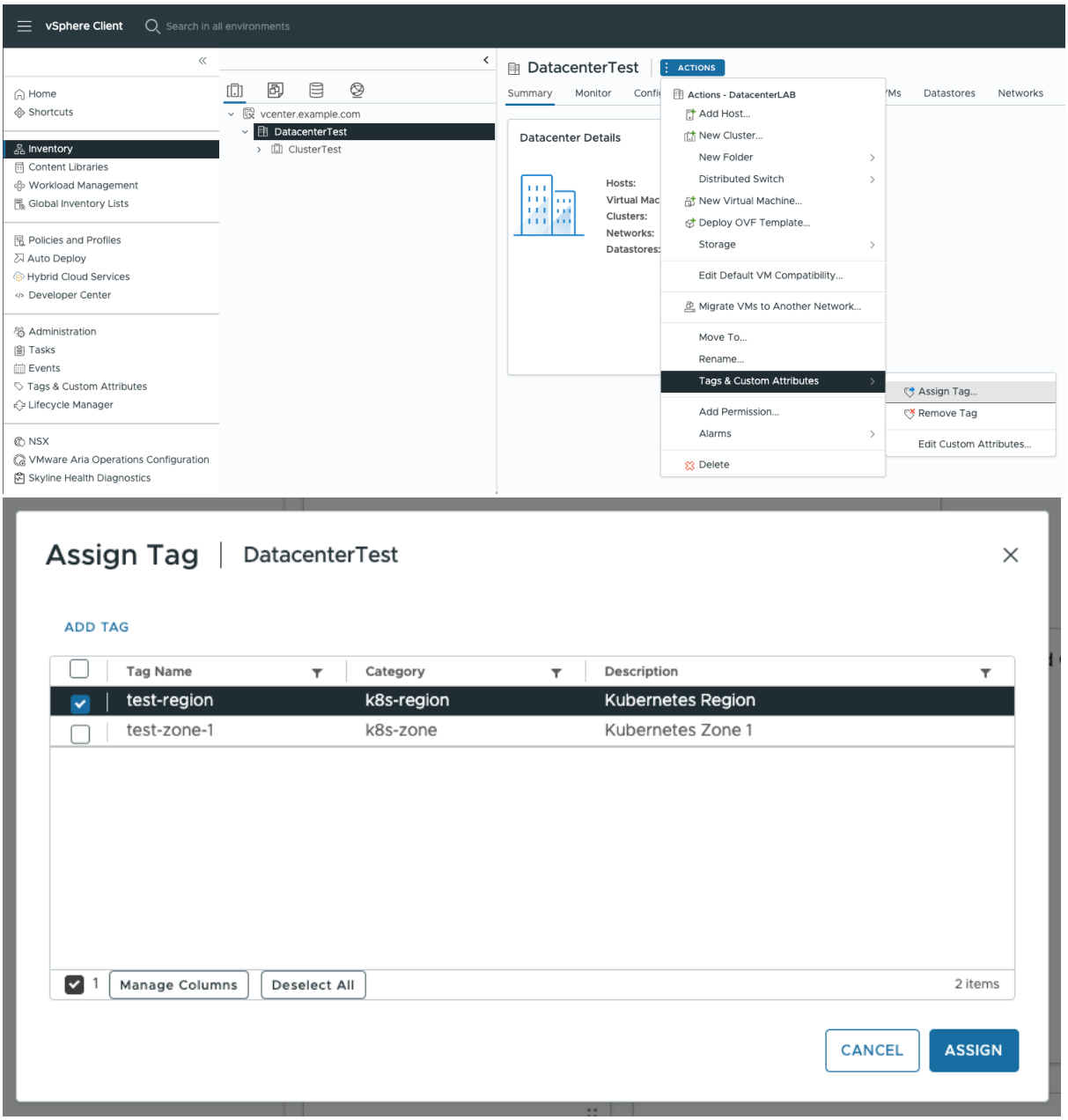
3. Create a second category for zones (for example, `k8s-zone`) with the object types Host, Cluster, and Datastore.



4. Go to the “Tags” tab and create at least one tag in the region category and one tag in the zone category (for example, `test-region`, `test-zone-1`).



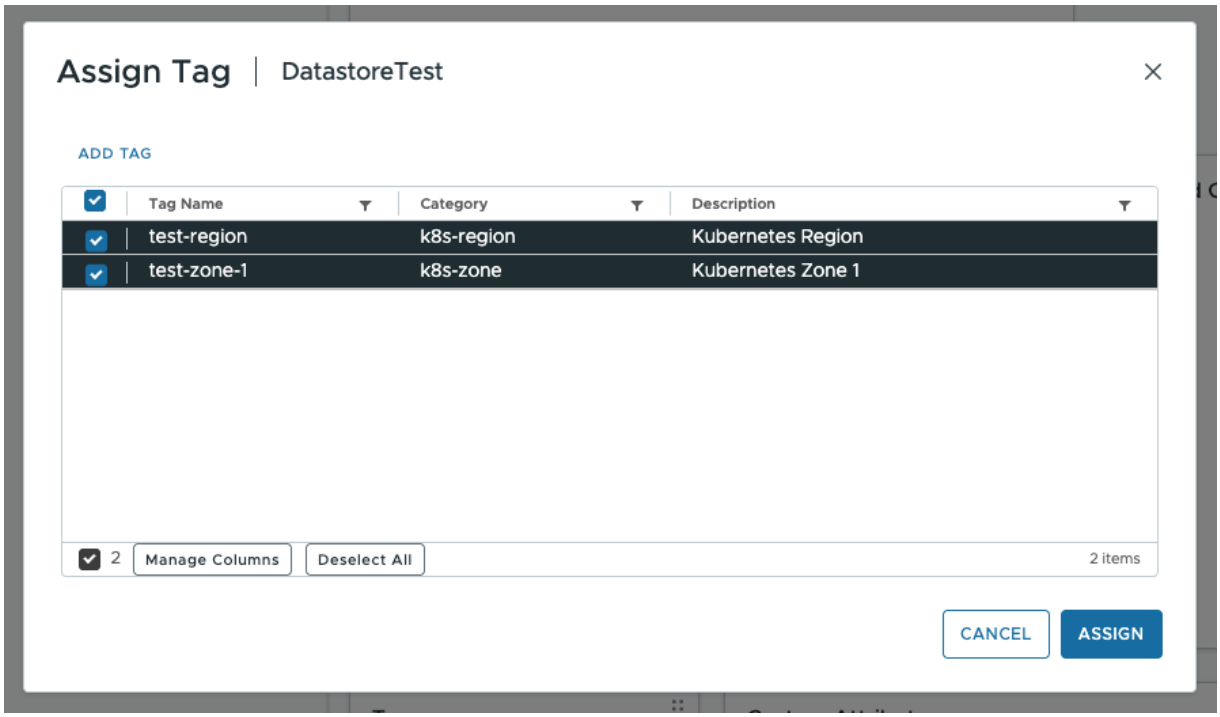
5. In the “Inventory” tab, select the target Datacenter, open the “Summary” panel, then choose “Actions” → “Tags & Custom Attributes” → “Assign Tag” and assign the region tag. Repeat this step for each Cluster that will host nodes, assigning the appropriate zone tags.



Configuring Datastore in vSphere Client

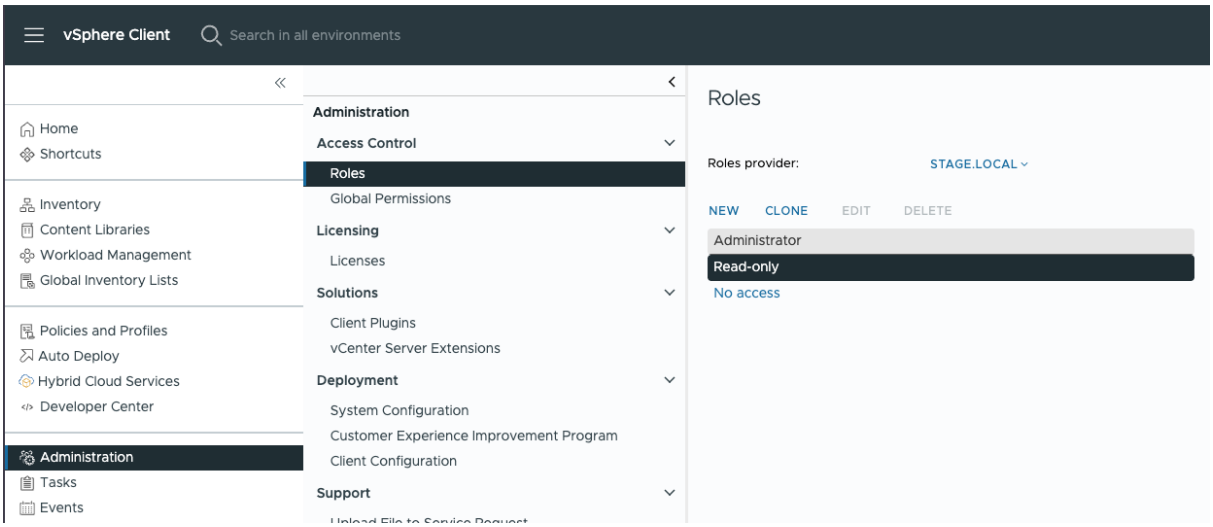
> ⚠ For dynamic provisioning of PersistentVolume, the Datastore must be available on **every** ESXi host in the zone (shared datastore).

In the “Inventory” tab, select the Datastore, open the “Summary” panel, then choose “Actions” → “Tags & Custom Attributes” → “Assign Tag”. Assign the Datastore the same region tag as the corresponding Datacenter, and the same zone tag as the corresponding Cluster.

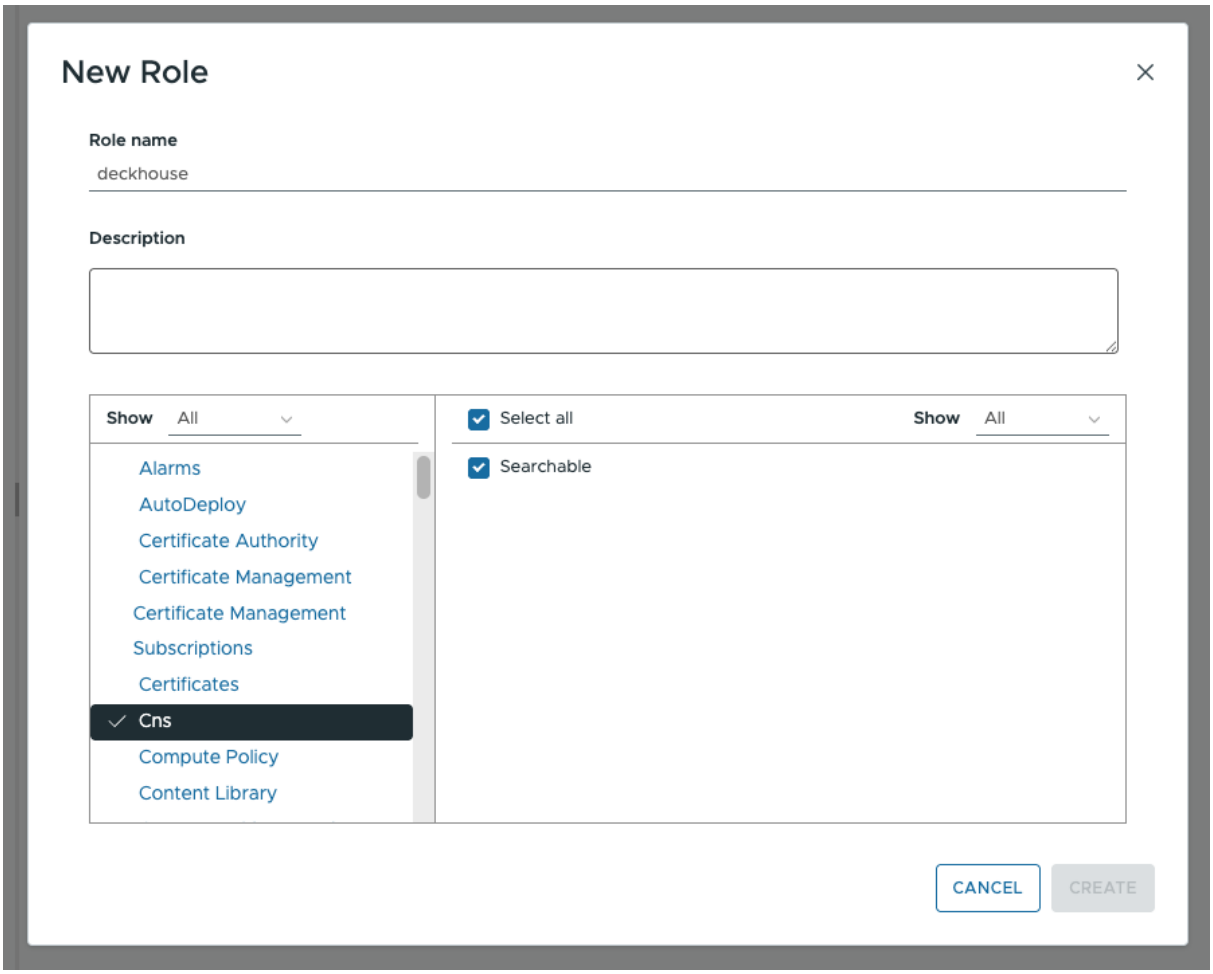


Creating and assigning a role in vSphere Client

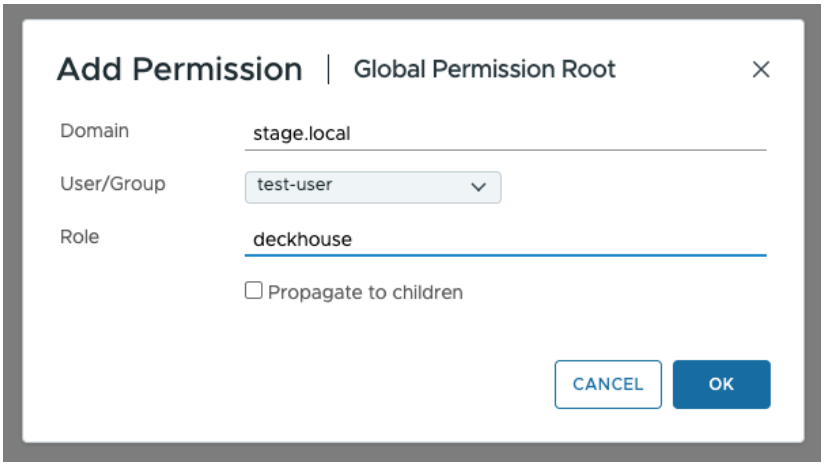
1. Go to “Menu” → “Administration” → “Access Control” → “Roles”.



2. Click “NEW”, enter a role name (for example, `deckhouse`), and add the privileges from the list.



3. Assign the role to the Deckhouse service account: go to “Menu” → “Administration” → “Access Control” → “Global Permissions”, click “ADD”, and select the user and the `deckhouse` role.



Configuration via govc

Installing govc

To continue configuring vSphere, install the govc CLI utility.

After installation, set the environment variables required to connect to `vCenter`.

⚠ Make sure to specify the username together with the domain, for example: `username@domain.local`.

```
export GOVC_URL=example.com
export GOVC_USERNAME=<username>@vsphere.local
export GOVC_PASSWORD=<password>
export GOVC_INSECURE=1
```

Creating tags and tag categories with govc

VMware vSphere does not have built-in concepts of a “region” or a “zone”. In vSphere, a Datacenter acts as a region, and a Cluster acts as a zone. Tags are used to establish this mapping.

Create tag categories with the following commands:

```
govc tags.category.create -d "Kubernetes Region" k8s-region
govc tags.category.create -d "Kubernetes Zone" k8s-zone
```

Create tags in each category. If you intend to use multiple “zones” (Cluster), create a tag for each one of them:

```
govc tags.create -d "Kubernetes Region" -c k8s-region test-region
govc tags.create -d "Kubernetes Zone Test 1" -c k8s-zone test-zone-1
govc tags.create -d "Kubernetes Zone Test 2" -c k8s-zone test-zone-2
```

Attach the “region” tag to Datacenter:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>
```

Attach “zone” tags to the Cluster objects:

```
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/host/<ClusterName1>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/host/<ClusterName2>
```

Datastore configuration with govc

⚠ For dynamic PersistentVolume provisioning, a Datastore must be available on **each** ESXi host (shared datastore).

Assign the “region” and “zone” tags to the Datastore objects to automatically create a StorageClass in the Kubernetes cluster:

```
govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName1>
govc tags.attach -C k8s-zone test-zone-1 /<DatacenterName>/datastore/<DatastoreName1>

govc tags.attach -C k8s-region test-region /<DatacenterName>/datastore/<DatastoreName2>
govc tags.attach -C k8s-zone test-zone-2 /<DatacenterName>/datastore/<DatastoreName2>
```

Creating and assigning a role with govc

i We’ve intentionally skipped User creation since there are many ways to authenticate a user in the vSphere.

The role described below includes the privileges from the list of required privileges section. If you need a more granular Role, please contact your Deckhouse support.

Create a role with the required privileges:

```
govc role.create deckhouse \
    Cns.Searchable \
    Datastore.AllocateSpace Datastore.Browse Datastore.FileManagement \
    Folder.Create Folder.Delete Folder.Move Folder.Rename \
    Global.GlobalTag Global.SystemTag \
    InventoryService.Tagging.AttachTag InventoryService.Tagging.CreateCategory \
    InventoryService.Tagging.CreateTag InventoryService.Tagging.DeleteCategory \
    InventoryService.Tagging.DeleteTag InventoryService.Tagging.EditCategory \
    InventoryService.Tagging.EditTag InventoryService.Tagging.ModifyUsedByForCategory \
    InventoryService.Tagging.ModifyUsedByForTag InventoryService.Tagging.ObjectAttachable \
    Network.Assign \
    Resource.AssignVMToPool Resource.CreatePool Resource.DeletePool Resource.EditPool Resource.RenamePool \
    StorageProfile.View \
    System.Anonymous System.Read System.View \
    VApp.ApplicationConfig VApp.AssignResourcePool VApp.AssignVM VApp.Create VApp.Delete \
    VApp.ExtractOvfEnvironment VApp.Import VApp.InstanceConfig VApp.PowerOff VApp.PowerOn VApp.ResourceConfig \
    VirtualMachine.Config.AddExistingDisk VirtualMachine.Config.AddNewDisk VirtualMachine.Config.AddRemoveDevice \
    VirtualMachine.Config.AdvancedConfig VirtualMachine.Config.Annotation VirtualMachine.Config.CPUCount \
    VirtualMachine.Config.ChangeTracking VirtualMachine.Config.DiskExtend VirtualMachine.Config.DiskLease \
    VirtualMachine.Config.EditDevice VirtualMachine.Config.ManagedBy VirtualMachine.Config.Memory \
    VirtualMachine.Config.QueryUnownedFiles VirtualMachine.Config.RawDevice VirtualMachine.Config.ReloadFromPath \
    VirtualMachine.Config.RemoveDisk VirtualMachine.Config.Rename VirtualMachine.Config.ResetGuestInfo \
    VirtualMachine.Config.Resource VirtualMachine.Config.Settings VirtualMachine.Config.SwapPlacement \
    VirtualMachine.Config.UpgradeVirtualHardware \
    VirtualMachine.GuestOperations.Query \
    VirtualMachine.Interact.AnswerQuestion VirtualMachine.Interact.DeviceConnection \
    VirtualMachine.Interact.GuestControl VirtualMachine.Interact.PowerOff VirtualMachine.Interact.PowerOn \
    VirtualMachine.Interact.Reset VirtualMachine.Interact.SetCDMedia VirtualMachine.Interact.ToolsInstall \
    VirtualMachine.Inventory.Create VirtualMachine.Inventory.CreateFromExisting VirtualMachine.Inventory.Delete \
    VirtualMachine.Inventory.Move \
    VirtualMachine.Provisioning.Clone VirtualMachine.Provisioning.Customize VirtualMachine.Provisioning.DeployTemplate \
    VirtualMachine.Provisioning.GetVmFiles VirtualMachine.Provisioning.PutVmFiles VirtualMachine.Provisioning.ReadCustSpecs
\
    VirtualMachine.State.CreateSnapshot VirtualMachine.State.RemoveSnapshot VirtualMachine.State.RenameSnapshot
```

Assign the role to a user on the vCenter object.

⚠ Make sure to specify the username together with the domain, for example: `username@domain.local`.

```
govc permissions.set -principal <username>@vsphere.local -role deckhouse /
```

i For more detailed permission configuration, refer to the official documentation.

VM image requirements

To create a VM template (`Template`), it is recommended to use a ready-made cloud image/OVA file provided by the OS vendor:

- **Ubuntu**
- **Debian**
- **CentOS**
- **Rocky Linux** (section *Generic Cloud / OpenStack*)

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Preparing the virtual machine image

⚠ Disable VMware Guest OS Customization (and any vApp/OS customization mechanisms, if applicable in your setup) for the template and the cluster virtual machines. DKP performs the initial node configuration via `cloud-init` (VMware GuestInfo datasource). Enabled customization can conflict with `cloud-init` and lead to incorrect node initialization.

1. Install the required packages:

   If you use `cloud-init` version lower than 21.3 (VMware GuestInfo support is required):

```
sudo apt-get update
sudo apt-get install -y open-vm-tools cloud-init cloud-init-vmware-guestinfo
```

If you use `cloud-init` version 21.3 or higher:

```
sudo apt-get update
sudo apt-get install -y open-vm-tools cloud-init
```

2. Verify that the `disable_vmware_customization: false` parameter is set in `/etc/cloud/cloud.cfg`.
3. Make sure the `default_user` parameter is specified in `/etc/cloud/cloud.cfg`. It is required to add an SSH key when the VM starts.
4. Add the VMware GuestInfo datasource — create `/etc/cloud/cloud.cfg.d/99-DataSourceVMwareGuestInfo.cfg`:

```
datasource:
  VMware:
    vmware_cust_file_max_wait: 10
```

5. Before creating the VM template, reset the `cloud-init` identifiers and state using the following commands:

```
truncate -s 0 /etc/machine-id &&
rm /var/lib/dbus/machine-id &&
ln -s /etc/machine-id /var/lib/dbus/machine-id
```

6. Clear `cloud-init` event logs:

```
cloud-init clean --logs --seed
```

⚠ After the virtual machine starts, the following services related to the packages installed during `cloud-init` preparation must be running:

- `cloud-config.service`,
- `cloud-final.service`,
- `cloud-init.service`.

To ensure that the services are enabled, use the command:

```
systemctl is-enabled cloud-config.service cloud-init.service cloud-final.service
```

Example output for enabled services:

```
enabled
enabled
enabled
```

i DKP creates VM disks of type `eagerZeroedThick`, but the type of disks of created VMs may be changed without notification according to the `VM Storage Policy` settings in vSphere.
For more details, see the documentation.

i DKP uses the `ens192` interface as the default interface for VMs in vSphere. Therefore, when using static IP addresses in `mainNetwork`, you must create an interface named `ens192` in the OS image as the default interface.

Infrastructure

Networking

A VLAN with DHCP and Internet access is required for the running cluster:

- If the VLAN is public (public addresses), then you have to create a second network to deploy cluster nodes (DHCP is not needed in this network).
- If the VLAN is private (private addresses), then this network can be used for cluster nodes.

Inbound traffic

- You can use an internal load balancer (if present) and direct traffic directly to the front nodes of the cluster.
- If there is no load balancer, you can use MetalLB in BGP mode to organize fault-tolerant load balancers (recommended). In this case, front nodes of the cluster will have two interfaces. For this, you will need:
  - A dedicated VLAN for traffic exchange between BGP routers and MetalLB. This VLAN must have DHCP and Internet access.
  - IP addresses of BGP routers.

- ASN — the AS number on the BGP router.
- ASN — the AS number in the cluster.
- A range to announce addresses from.

Using the datastore

Various types of storage can be used in the cluster; for the minimum configuration, you will need:

- Datastore for provisioning PersistentVolumes to the Kubernetes cluster.
- Datastore for provisioning root disks for the VMs (it can be the same Datastore as for PersistentVolume).

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

VsphereInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of a group of vSphere VirtualMachines used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalNetworks**
    array of strings

    Paths to networks that VirtualMachines' secondary NICs will connect to. Relative to the datacenter.

    Examples:

    ```
    additionalNetworks: DEVOPS_32
    ```

    ```
    additionalNetworks: DEVOPS_50
    ```

  - **spec.datastore**
    string

    Path to a Datastore (relative to the datacenter) in which VirtualMachines will be cloned.

    By default, a datastore from the `master` InstanceClass is used.

    Example:

    ```
    datastore: lun-1201
    ```

  - **spec.disableTimesync**
    boolean

    Disable time synchronization in Guest VM.

    > **i** **Note:** this won't disable any NTP daemons running on the guest OS, it only disables time synchronization from ESXi.

    Default: `true`

  - **spec.mainNetwork**
    string

    Path to the network that VirtualMachines' primary NICs will connect to (default gateway). Relative to the datacenter.

    If no value is specified, the value specified in `VsphereClusterConfiguration` will be used.

    Example:

    ```
    mainNetwork: k8s-msk-178
    ```

  - **spec.memory**

integer

Required value

Memory in MiB to allocate to vSphere VirtualMachines.

Example:

```
memory: 8192
```

- **spec.numCPUs**
integer

  Required value

  Count of vCPUs to allocate to vSphere VirtualMachines.

  Example:

  ```
  numCPUs: 2
  ```

- **spec.resourcePool**
string

  Path to a Resource Pool in which VirtualMachines will be cloned. Relative to the zone (vSphere Cluster).

  By default, the value from `VsphereCloudDiscoveryData` is used.

  Example:

  ```
  resourcePool: rp-2012
  ```

- **spec.rootDiskSize**
integer

  Root disk size in GiB to use in vSphere VirtualMachines.

  The disk will be automatically enlarged if its size in the template is less than specified.

  Default: `20`

  Example:

  ```
  rootDiskSize: 20
  ```

- **spec.runtimeOptions**
object

  Additional VM’s parameters.

  - **spec.runtimeOptions.cpuLimit**
  integer

    CPU limit in MHz.

  - **spec.runtimeOptions.cpuReservation**
  integer

    CPU reservation in MHz.

  - **spec.runtimeOptions.cpuShares**
  integer

    The relative amount of CPU Shares for VMs to be created.

  - **spec.runtimeOptions.memoryLimit**
  integer

    Memory limit in MB.

  - **spec.runtimeOptions.memoryReservation**
  integer

    VM memory reservation in percent (relative to `.spec.memory`).

    Default: `80`

    Allowed values: `0 <= X <= 100`

  - **spec.runtimeOptions.memoryShares**
  integer

    The relative amount of Memory Shares for VMs to be created.

    Allowed values: `0 <= X <= 100`

  - **spec.runtimeOptions.nestedHardwareVirtualization**
  boolean

    Whether to enable or disable nested Hardware Assisted Virtualization.

  - **spec.template**
    string

    Path to the template to be cloned. Relative to the datacenter.

    By default, a template from the `master` InstanceClass is used.

    Example:

    ```
    template: dev/golden_image
    ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Parameters of a group of vSphere VirtualMachines used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalNetworks**
    array of strings

    Paths to networks that VirtualMachines' secondary NICs will connect to. Relative to the datacenter.

    Examples:

    ```
    additionalNetworks: DEVOPS_32
    ```

    ```
    additionalNetworks: DEVOPS_50
    ```

  - **spec.datastore**
    string

    Path to a Datastore (relative to the datacenter) in which VirtualMachines will be cloned.

    By default, a datastore from the `master` InstanceClass is used.

    Example:

    ```
    datastore: lun-1201
    ```

  - **spec.disableTimesync**
    boolean

    Disable time synchronization in Guest VM.

    > **i** **Note:** this won't disable any NTP daemons running on the guest OS, it only disables time synchronization from ESXi.

    Default: `true`

  - **spec.mainNetwork**
    string

    Path to the network that VirtualMachines' primary NICs will connect to (default gateway). Relative to the datacenter.

    If no value is specified, the value specified in `VsphereClusterConfiguration` will be used.

    Example:

    ```
    mainNetwork: k8s-msk-178
    ```

  - **spec.memory**
    integer

    Required value

    Memory in MiB to allocate to vSphere VirtualMachines.

    Example:

    ```
    memory: 8192
    ```

  - **spec.numCPUs**
    integer

    Required value

Count of vCPUs to allocate to vSphere VirtualMachines.

Example:

```
numCPUs: 2
```

- **spec.resourcePool**
  string

  Path to a Resource Pool in which VirtualMachines will be cloned. Relative to the zone (vSphere Cluster).

  By default, the value from `VsphereCloudDiscoveryData` is used.

  Example:

  ```
  resourcePool: rp-2012
  ```

- **spec.rootDiskSize**
  integer

  Root disk size in GiB to use in vSphere VirtualMachines.

  The disk will be automatically enlarged if its size in the template is less than specified.

  Default: `20`

  Example:

  ```
  rootDiskSize: 20
  ```

- **spec.runtimeOptions**
  object

  Additional VM’s parameters.

  - **spec.runtimeOptions.cpuLimit**
    integer

    CPU limit in MHz.

  - **spec.runtimeOptions.cpuReservation**
    integer

    CPU reservation in MHz.

  - **spec.runtimeOptions.cpuShares**
    integer

    The relative amount of CPU Shares for VMs to be created.

  - **spec.runtimeOptions.memoryLimit**
    integer

    Memory limit in MB.

  - **spec.runtimeOptions.memoryReservation**
    integer

    VM memory reservation in percent (relative to `.spec.memory`).

    Default: `80`

    Allowed values: `0 <= X <= 100`

  - **spec.runtimeOptions.memoryShares**
    integer

    The relative amount of Memory Shares for VMs to be created.

    Allowed values: `0 <= X <= 100`

  - **spec.runtimeOptions.nestedHardwareVirtualization**
    boolean

    Whether to enable or disable nested Hardware Assisted Virtualization.

- **spec.template**
  string

  Path to the template to be cloned. Relative to the datacenter.

  By default, a template from the `master` InstanceClass is used.

  Example:

  ```
  template: dev/golden_image
  ```

- **status**
  object

- **status.nodeGroupConsumers**
  array of strings

Examples

> **i** The module lifecycle stage: General Availability

Below is an example configuration for a VMware vSphere cloud provider.

An example of the configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-vsphere
spec:
  version: 2
  enabled: true
  settings:
    host: vc-3.internal
    username: user
    password: password
    vmFolderPath: dev/test
    insecure: true
    region: moscow-x001
    sshKeys:
    - "ssh-rsa AAAAB3N....6xHJwwj"
    externalNetworkNames:
    - KUBE-3
    - devops-internal
    internalNetworkNames:
    - KUBE-3
    - devops-internal
    nsxt:
      defaultIpPoolName: "External IP Pool"
      tier1GatewayPath: flant_tier1
      user: guestuser1
      password: pass
      host: 1.2.3.4
      insecureFlag: true
      size: SMALL
```

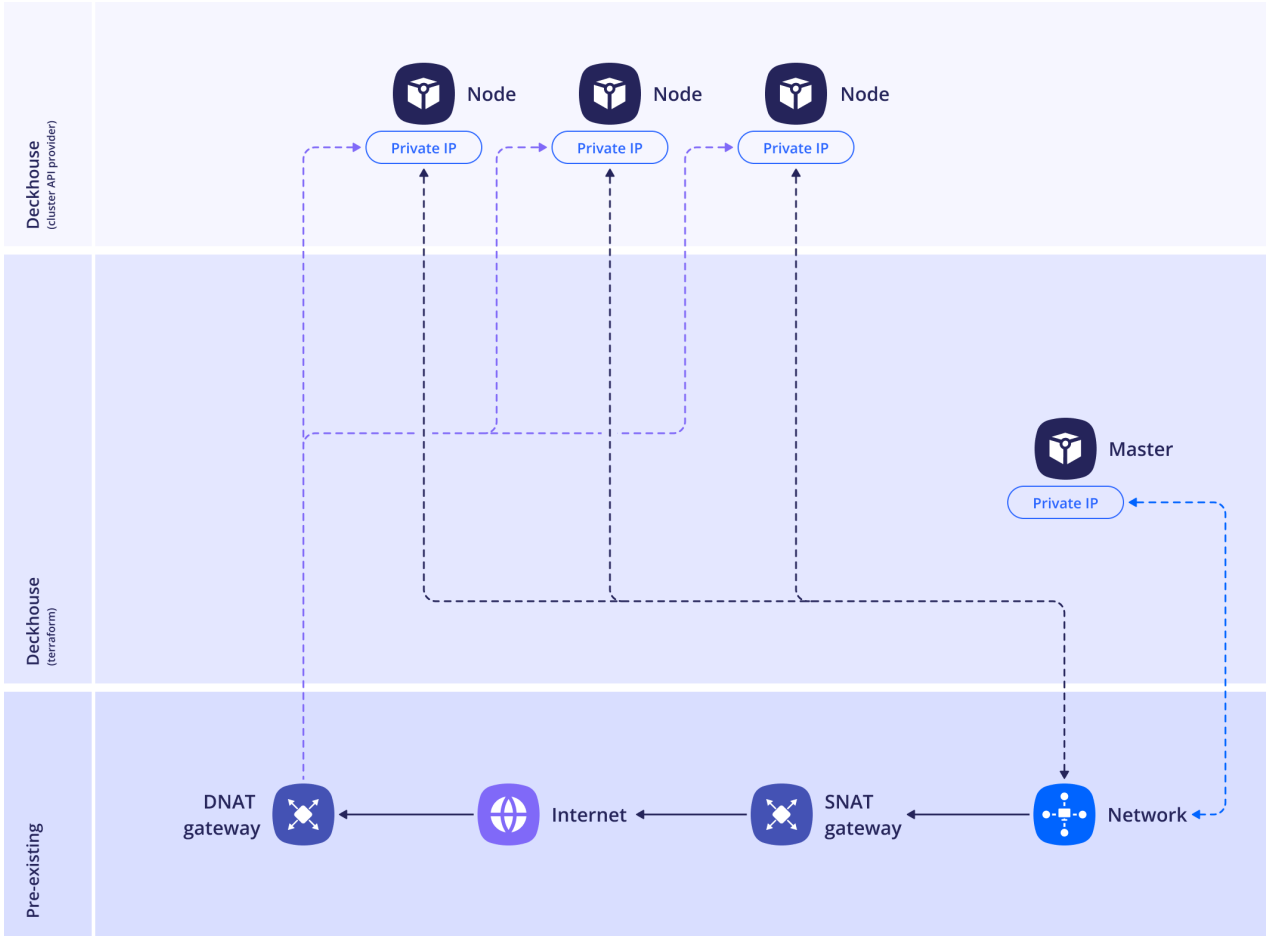
An example of the `VsphereInstanceClass` custom resource

```
apiVersion: deckhouse.io/v1
kind: VsphereInstanceClass
metadata:
  name: test
spec:
  numCPUs: 2
  memory: 2048
  rootDiskSize: 20
  template: dev/golden_image
  mainNetwork: k8s-msk-178
  datastore: <DS_PARENT_FOLDER>/lun-1201 # <DS_PARENT_FOLDER> — parent Datastore folder.
```

Cloud provider - VMware vSphere: Layouts

> **i** The module lifecycle stage: General Availability

Standard



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: VsphereClusterConfiguration
layout: Standard
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
vmFolderPath: dev
regionTagCategory: k8s-region
zoneTagCategory: k8s-zone
region: X1
internalNetworkCIDR: 192.168.199.0/24
masterNodeGroup:
  replicas: 1
  zones:
  - ru-central1-a
  - ru-central1-b
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: dev/golden_image
    datastore: dev/lun_1
    mainNetwork: net3-k8s
nodeGroups:
- name: khm
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: dev/golden_image
    datastore: dev/lun_1
    mainNetwork: net3-k8s
sshPublicKey: "<SSH_PUBLIC_KEY>"
zones:
- ru-central1-a
- ru-central1-b
```

Cloud provider — Yandex Cloud

> **i** The module lifecycle stage: General Availability

The `cloud-provider-yandex` module integrates Deckhouse Kubernetes Platform with Yandex Cloud. It allows the `node-manager` module to use Yandex Cloud resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-yandex` module:

- Managing Yandex Cloud resources via `cloud-controller-manager`:
  - creates network routes for the `PodNetwork` network on the Yandex Cloud side;
  - creates Network Load Balancers and target groups for Services of the LoadBalancer type;
  - updates instance and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in Yandex Cloud.
- Provisioning disks via the Yandex CSI driver (`yandex.csi.flant.com`) and creating StorageClasses for Yandex Cloud disk types so that PersistentVolumes can be requested from the cluster.
- Provisioning base infrastructure and CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-yandex`.
- Provisioning CloudEphemeral nodes via Machine Controller Manager (MCM) or Cluster API (CAPI). Virtual machine parameters are set in the YandexInstanceClass resource.
- Registering with `node-manager` so that YandexInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. Starting with DKP 1.76, `cni-cilium` is used by default.

> **⚠** The module is migrating CloudEphemeral node management from Machine Controller Manager (MCM) to Cluster API (CAPI). Existing NodeGroups continue to use MCM, while new ones are created with CAPI by default. For migrating existing groups, see How to migrate node groups to Cluster API (CAPI).

Configuration

> **i** The module lifecycle stage: General Availability

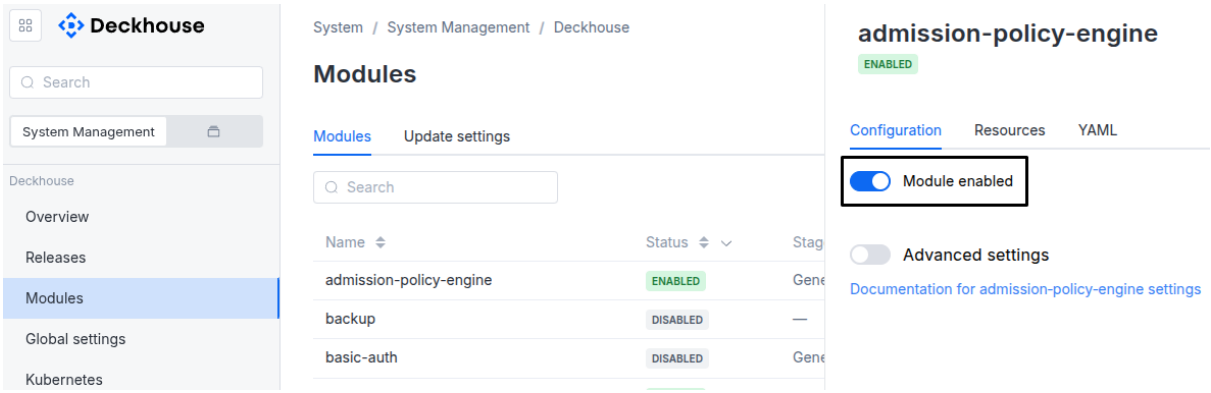
> **Note!** If the parameters provided below are changed, the **existing Machines are NOT redeployed** (new machines will be created with the updated parameters). Redeployment is only performed when `NodeGroup` and `YandexInstanceClass` are changed. Details in the node-manager module's documentation.

The module has 3 alerts.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `cloud-provider-yandex` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cloud-provider-yandex
```

- **Using ModuleConfig `cloud-provider-yandex`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-yandex` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-yandex`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  enabled: true
```

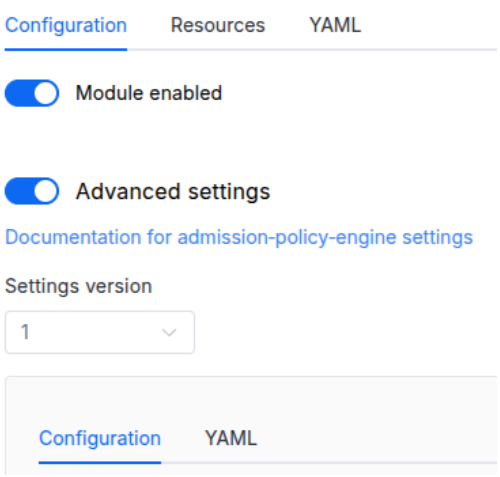
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `cloud-provider-yandex` module and enable the "Advanced Settings" switch. Fill in the required fields in the "Configuration" tab or specify the module settings in YAML format on the "YAML" tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `cloud-provider-yandex` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-yandex`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-yandex` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-yandex
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cloud-provider-yandex` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cloud-provider-yandex`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-yandex
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Storage

The module automatically creates StorageClasses covering all available disks in Yandex:

Type	StorageClass Name	Comment
network-hdd	network-hdd	
network-ssd	network-ssd	
network-ssd-nonreplicated	network-ssd-nonreplicated	
network-ssd-io-m3	network-ssd-io-m3	Disk size must be a multiple of 93 GB.

You can filter out the unnecessary StorageClasses via the exclude parameter.

LoadBalancer

The module subscribes to Service objects of the `LoadBalancer` type and creates the corresponding `NetworkLoadBalancer` and `TargetGroup` in Yandex Cloud.

For more information, see the Kubernetes Cloud Controller Manager for Yandex Cloud documentation.

Parameters

Schema version: 1

- **settings**
  object
  - **settings.additionalExternalNetworkIDs**
    array of strings

    A list of Network IDs that will be considered `ExternalIP` when listing Node addresses.

    Default: `[]`

  - **settings.storageClass**
    object
    - **settings.storageClass.default**
      Deprecated
      string

      The name of StorageClass that will be used in the cluster by default.

      If the parameter is omitted, the default StorageClass will be one of the following:

      - an arbitrary StorageClass present in the cluster that has the default annotation;
      - the first (in lexicographic order) StorageClass created by the module.

> **Parameter is deprecated.** Instead, use the global parameter global.defaultClusterStorageClass.

Example:

```
default: network-hdd
```

- **settings.storageClass.exclude**
  array of strings

  A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster;

  Example:

```
exclude:
- network-ssd-.*
- network-hdd
```

Provider configuration

> The module lifecycle stage: General Availability

YandexClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in Yandex Cloud.

Used by the cloud provider if a cluster’s control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

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

> After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: Standard
nodeNetworkCIDR: 127.0.0.1/8
labels:
  label-2: b
sshPublicKey: "<SSH_PUBLIC_KEY>"
masterNodeGroup:
  replicas: 1
  instanceClass:
    cores: 4
    memory: 8192
    imageID: fd83ica41cade1mj35sr
nodeGroups:
- name: worker
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    cores: 4
    memory: 8192
    imageID: fd83ica41cade1mj35sr
    coreFraction: 50
    externalIPAddresses:
    - 198.51.100.5
    - Auto
provider:
  cloudID: "<CLOUD_ID>"
  folderID: "<FOLDER_ID>"
  serviceAccountJSON: |
    {
    "id": "id",
    "service_account_id": "service_account_id",
    "key_algorithm": "RSA_2048",
    "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID....AQAB\n-----END PUBLIC KEY-----\n",
    "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE....1ZPJeBLt+\n-----END PRIVATE KEY-----\n"
    }
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`, `deckhouse.io/v1alpha1`

- **dhcpOptions**
  object

  A list of DHCP parameters to use for all subnets.

  Note that setting dhcpOptions may lead to problems.

  - **dhcpOptions.domainName**
    string

    The name of the search domain.

  - **dhcpOptions.domainNameServers**
    array of strings

    A list of recursive DNS addresses.

    - **Element of the array**
      string

      Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

- **existingNetworkID**
  string

  The ID of the existing VPC Network.

- **existingZoneToSubnetIDMap**
  object

  One or more pre-existing subnets mapped to respective zone.

  Warning. When using `cni-simple-bridge`, DKP creates a route table that must be manually associated with the specified subnets. Only one route table can be associated with a subnet, so multiple clusters using `cni-simple-bridge` cannot be deployed in the same subnets. Starting with DKP 1.76, new clusters in Yandex Cloud use `cni-cilium` in `VXLAN` mode by default. In this mode, pod traffic routing does not depend on Yandex Cloud route tables.

Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

Example:

```
existingZoneToSubnetIDMap:
  ru-central1-a: e2lu8r1tbbtryhdpa9ro
  ru-central1-b: e2lu8r1tbbtryhdpa9ro
  ru-central1-d: e2lu8r1tbbtryhdpa9ro
```

- **kind**
  string

  Required value

  Allowed values: `YandexClusterConfiguration`

- **labels**
  object

  Labels to attach to resources created in the Yandex Cloud.

  Note that you have to re-create all the machines to add new labels if labels were modified in the running cluster.

- **layout**
  string

  Required value

  The way resources are located in the cloud.

  Read more about possible provider layouts.

  Allowed values: `Standard`, `WithoutNAT`, `WithNATInstance`

- **masterNodeGroup**
  object

  Required value

  The definition of the master's NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the fields of the YandexInstanceClass.

    - **masterNodeGroup.instanceClass.additionalLabels**
      object

      Additional labels.

      Example:

      ```
      project: cms-production
      severity: critical
      ```

    - **masterNodeGroup.instanceClass.cores**
      integer

      Required value

      Amount of CPU cores to provision on a Yandex Compute Instance.

    - **masterNodeGroup.instanceClass.diskSizeGB**
      integer

      Yandex Compute Instance disk size in gibibytes.

      Default: `50`

      Example:

      ```
      diskSizeGB: 0
      ```

    - **masterNodeGroup.instanceClass.diskType**
      string

      Yandex Compute Instance disk type. Size of `network-ssd-nonreplicated` and `network-ssd-io-m3` disks must be a multiple of 93 GB.

      Default: `network-ssd`

Allowed values: `network-ssd`, `network-ssd-io-m3`, `network-ssd-nonreplicated`

Example:

```
diskType: network-ssd-io-m3
```

- **masterNodeGroup.instanceClass.etcdDiskSizeGb**
  integer

  etcd disk size in gibibytes.

  If this parameter is changed, each etcd disk must be manually expanded in the Yandex Cloud interface (the reason for this behavior is in the issue).

  Default: `10`

  Example:

  ```
  etcdDiskSizeGb: 10
  ```

- **masterNodeGroup.instanceClass.externalIPAddresses**
  array of strings

  A list of external IP addresses, listed in the order of the zones where nodes will be created.

  The following values can be specified in the list:

  - IP address from an additional external network for the corresponding zone (parameter `externalSubnetIDs`);
  - reserved public IP address, if the list of additional external networks is not defined (parameter `externalSubnetIDs`);
  - `Auto`, to order a public IP address in the corresponding zone.

  Please note:

  - The array elements must be listed according to the order of zones (parameter `zones`) in which the nodes will be created.

    For example:

    ```
    - <RU-CENTRAL1-A-IP-ADDRESS>
    - <RU-CENTRAL1-B-IP-ADDRESS>
    - <RU-CENTRAL1-D-IP-ADDRESS>
    ```

  - The number of elements in the `externalIPAddresses` array must equal the number of nodes in the group (the `replicas` parameter). To automatically order public IP addresses in the zone, use the value `Auto` in the list of the `externalIPAddresses` parameter at the corresponding position (matching the order of zones).
  - If `externalIPAddresses` is not specified, the nodes in the group will be ordered without external IP addresses.
  - **Element of the array**
    string

    Pattern: `^([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})|(Auto)$`

- **masterNodeGroup.instanceClass.externalSubnetID**
  Deprecated
  string

  If specified, an additional network interface will be added to the node (the latter will use it as a default route).

- **masterNodeGroup.instanceClass.externalSubnetIDs**
  array of strings

  List of additional external networks connected to the nodes.

  The node interface that is in the list will be assigned as the default gateway.

  Also, a route for the node's internal network interface will be added for the entire subnet specified in nodeNetworkCIDR.

  The array elements **must** be listed in the same order as the zones (parameter `zones`) in which the nodes will be created. For example:

  ```
  - <RU-CENTRAL1-A-SUBNET-ID>
  - <RU-CENTRAL1-B-SUBNET-ID>
  - <RU-CENTRAL1-D-SUBNET-ID>
  ```

- **masterNodeGroup.instanceClass.imageID**
  string

  Required value

  Image ID to use while provisioning Yandex Compute Instances.

  Example:

```
imageID: fd83ica41cade1mj35sr
```

- **masterNodeGroup.instanceClass.memory**
  integer

  Required value

  Amount of primary memory in MB provision on a Yandex Compute Instance.

  Example:

  ```
  memory: 8192
  ```

- **masterNodeGroup.instanceClass.networkType**
  string

  Network type.

  Default: `Standard`

  Allowed values: `Standard`, `SoftwareAccelerated`

- **masterNodeGroup.instanceClass.platform**
  string

  The type of virtual machine to create.

  Default: `standard-v2`

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **masterNodeGroup.zones**
  array

  Not required value.

  A limited set of zones in which master nodes can be created.

  The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

  - **Element of the array**

    Allowed values: `ru-central1-a`, `ru-central1-b`, `ru-central1-d`

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the YandexInstanceClass.

    - **nodeGroups.instanceClass.additionalLabels**
      object

      Additional labels.

      Example:

      ```
      project: cms-production
      severity: critical
      ```

    - **nodeGroups.instanceClass.coreFraction**
      integer

      Percent of reserved CPU capacity on a Yandex Compute Instance. Details...

      Default: `100`

      Allowed values: `5`, `20`, `50`, `100`

      Example:

      ```
      coreFraction: 20
      ```

- **nodeGroups.instanceClass.cores**
  integer

  Required value

  Amount of CPU cores to provision on a Yandex Compute Instance.

- **nodeGroups.instanceClass.diskSizeGB**
  integer

  Yandex Compute Instance disk size in gibibytes.

  Default: `50`

  Example:

  ```
  diskSizeGB: 0
  ```

- **nodeGroups.instanceClass.diskType**
  string

  Yandex Compute Instance disk type. Size of `network-ssd-nonreplicated` and `network-ssd-io-m3` disks must be a multiple of 93 GB.

  Default: `network-ssd`

  Allowed values: `network-ssd`, `network-ssd-io-m3`, `network-ssd-nonreplicated`

  Example:

  ```
  diskType: network-ssd-io-m3
  ```

- **nodeGroups.instanceClass.externalIPAddresses**
  array of strings

  A list of external IP addresses, listed in the order of the zones where nodes will be created.

  The following values can be specified in the list:

  - IP address from an additional external network for the corresponding zone (parameter `externalSubnetIDs`);
  - reserved public IP address, if the list of additional external networks is not defined (parameter `externalSubnetIDs`);
  - `Auto`, to order a public IP address in the corresponding zone.

  Please note:

  - The array elements must be listed according to the order of zones (parameter `zones`) in which the nodes will be created.

    For example:

    ```
    - <RU-CENTRAL1-A-IP-ADDRESS>
    - <RU-CENTRAL1-B-IP-ADDRESS>
    - <RU-CENTRAL1-D-IP-ADDRESS>
    ```

  - The number of elements in the `externalIPAddresses` array must equal the number of nodes in the group (the `replicas` parameter). To automatically order public IP addresses in the zone, use the value `Auto` in the list of the `externalIPAddresses` parameter at the corresponding position (matching the order of zones).
  - If `externalIPAddresses` is not specified, the nodes in the group will be ordered without external IP addresses.
  - **Element of the array**
    string

    Pattern: `^([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})|(Auto)$`

- **nodeGroups.instanceClass.externalSubnetID**
  Deprecated
  string

  If specified, an additional network interface will be added to the node (the latter will use it as a default route).

- **nodeGroups.instanceClass.externalSubnetIDs**
  array of strings

  List of additional external networks connected to the nodes.

  The node interface that is in the list will be assigned as the default gateway.

  Also, a route for the node’s internal network interface will be added for the entire subnet specified in nodeNetworkCIDR.

  The array elements **must** be listed in the same order as the zones (parameter `zones`) in which the nodes will be created. For example:

```
- <RU-CENTRAL1-A-SUBNET-ID>
- <RU-CENTRAL1-B-SUBNET-ID>
- <RU-CENTRAL1-D-SUBNET-ID>
```

- **nodeGroups.instanceClass.imageID**
  string

  Required value

  Image ID to use while provisioning Yandex Compute Instances.

  Example:

  ```
  imageID: fd83ica41cade1mj35sr
  ```

- **nodeGroups.instanceClass.memory**
  integer

  Required value

  Amount of primary memory in MB provision on a Yandex Compute Instance.

  Example:

  ```
  memory: 8192
  ```

- **nodeGroups.instanceClass.networkType**
  string

  Network type.

  Default: `Standard`

  Allowed values: `Standard`, `SoftwareAccelerated`

- **nodeGroups.instanceClass.platform**
  string

  The type of virtual machine to create.

  Default: `standard-v2`

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: "true"
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to cluster resources.

    The same as the `metadata.labels` standard field.

    Note that you have to re-create all the machines to add new tags if tags were modified in the running cluster.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object.

> **i** Available fields: `effect`, `key`, and `values`.

Example:

```
taints:
- effect: NoExecute
  key: ship-class
  value: frigate
```

  - **nodeGroups.nodeTemplate.taints.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
  - **nodeGroups.nodeTemplate.taints.key**
    string
  - **nodeGroups.nodeTemplate.taints.value**
    string
- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes to create.
- **nodeGroups.zones**
  array

  A limited set of zones in which nodes can be created.

  The order in which zones are specified does not influence the sequence of node creation. Nodes are assigned to zones in alphabetical order: the first node is placed in the alphabetically first zone, the second node in the second zone, and so forth. If there are more nodes than zones, the assignment process restarts from the beginning of the zone list after placing a node in the last alphabetical zone.

  - **Element of the array**

    Allowed values: `ru-central1-a`, `ru-central1-b`, `ru-central1-d`

- **nodeNetworkCIDR**
  string

  Required value

  This subnet will be split into **three** equal parts.

  They will serve as a basis for subnets in three Yandex Cloud zones.
- **provider**
  object

  Required value

  Contains settings to connect to the Yandex Cloud API.

  - **provider.cloudID**
    string

    Required value

    The cloud ID.
  - **provider.folderID**
    string

    Required value

    ID of the directory.
  - **provider.serviceAccountJSON**
    string

    Required value

    A key to the Service Account in the JSON format.

    You can get it by using `[yc iam key create](environment.html)` command.

    Pattern: `^[ \t\n]*\{(.|\n)*\}[ \t\n]*$`

    Example:

```
serviceAccountJSON: |
 {
    "id": "...",
    "service_account_id": "...",
    "created_at": "2022-08-04T05:38:34.756137618Z",
    "key_algorithm": "RSA_2048",
    "public_key": "-----BEGIN PUBLIC KEY-----...-----END PUBLIC KEY-----\n",
    "private_key": "-----BEGIN PRIVATE KEY-----...-----END PRIVATE KEY-----\n"
 }
```

- **sshPublicKey**
  string

  Required value

  A public key for accessing nodes.

- **withNATInstance**
  object

  Settings for the `WithNATInstance` layout.

  - **withNATInstance.exporterAPIKey**
    string

    API-key for cloud metrics exporter.

    - If parameter is empty, cloud metrics exporter will not be deployed in the cluster.
    - If parameter is `Auto`, Deckhouse will create service account with the `monitoring.viewer` role and create API-key manually. Provider service account should have the `admin` role.
    - Any other value is considered a valid API-key. See this instruction for creating API-key. Service account should have `monitoring.viewer` role.

    Default:

  - **withNATInstance.externalSubnetID**
    string

    If specified, an additional network interface will be added to the node (the node will use it as a default route).

  - **withNATInstance.internalSubnetCIDR**

    CIDR of an automatically created subnet for the internal interface. Overrides `internalSubnetID` parameter.

  - **withNATInstance.internalSubnetID**
    string

    ID of a subnet for the internal interface.

  - **withNATInstance.natInstanceExternalAddress**
    string

    A reserved external IP address (or `externalSubnetID` address if specified).

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  - **withNATInstance.natInstanceInternalAddress**
    Deprecated
    string

    Consider using automatically generated address instead.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  - **withNATInstance.natInstanceResources**
    object

    Computing resources that are allocated to the NAT instance. If not specified, the default values will be used.

    > **i** **Warning.** If these parameters are changed, `terraform-auto-converger` will automatically restart NAT-instance if autoConvergerEnabled is set to `true`. This may result in a temporary interruption of network traffic in the cluster.

    - **withNATInstance.natInstanceResources.cores**
      integer

      Amount of CPU cores to provision on the NAT instance.

      Default: `2`

    - **withNATInstance.natInstanceResources.memory**
      integer

      Amount of primary memory in MB provision on the NAT instance.

      Default: `2048`

    - **withNATInstance.natInstanceResources.platform**

string

Processor platform type on the NAT instance.

Default: `standard-v2`

- **zones**
  array

  The globally restricted set of zones that this cloud provider works with.

  - **Element of the array**

    Allowed values: `ru-central1-a`, `ru-central1-b`, `ru-central1-d`

Preparing environment

ℹ The module lifecycle stage: General Availability

⚠ To use `ContainerdV2` as the container runtime on cluster nodes, the operating systems on virtual machines must meet the following requirements:

- Linux kernel version 5.8 or newer, except for the ranges 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS).
- `CgroupsV2` support.
- Systemd version `244` or newer.
- `erofs` kernel module support.

For more information, see the documentation.

⚠ The `cloud-init` package must be installed on the VMs. After the virtual machine is started, the following services associated with this package must be started:

- `cloud-config.service`;
- `cloud-final.service`;
- `cloud-init.service`.

⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

Creating a service account

You need to create a service account with the editor role with the cloud provider so that Deckhouse Kubernetes Platform (DKP) can manage cloud resources. The detailed instructions for creating a service account with Yandex Cloud are available in the provider's documentation. Below, we will provide a brief overview of the necessary actions:

1. Create a user named `deckhouse`:

   ```
   yc iam service-account create --name deckhouse
   ```

   The command output will contain its parameters:

   ```
   id: <userID>
   folder_id: <folderID>
   created_at: "YYYY-MM-DDTHH:MM:SSZ"
   name: deckhouse
   ```

2. Assign the required roles to the newly created user for your cloud:

   ```
   yc resource-manager folder add-access-binding --id <folderID> --role compute.editor --subject serviceAccount:
   <userID>
   yc resource-manager folder add-access-binding --id <folderID> --role vpc.admin --subject serviceAccount:<userID>
   yc resource-manager folder add-access-binding --id <folderID> --role load-balancer.editor --subject
   serviceAccount:<userID>
   ```

3. Create a JSON file containing the parameters for user authorization in the cloud. These parameters will be used to log in to the cloud:

   ```
   yc iam key create --service-account-name deckhouse --output deckhouse-sa-key.json
   ```

> **i** You may need to increase quotas.
>
> Reserve a public IP address if necessary.

Quotas

> **i** Note that you need to increase the quotas using the Yandex console when provisioning a new cluster.

Recommended quotas for a new cluster:

- The number of virtual processors: 64.
- The total volume of SSD disks: 2000 GB.
- The number of virtual machines: 25.
- The total amount of RAM of virtual machines: 256 GB.

Yandex Cloud integration

> **⚠** Starting with DKP version 1.76, Yandex Cloud uses the `cilium` CNI by default for new clusters.
>
> When using CentOS 8, you must explicitly enable the `cni-simple-bridge` module in the `config.yml` file.
>
> Example configuration for CentOS 8 in Yandex Cloud...
>
> ```
> apiVersion: deckhouse.io/v1alpha1
> kind: ModuleConfig
> metadata:
>  name: cni-simple-bridge
> spec:
>  enabled: true
> ```

Configuring security groups

When creating a cloud network, Yandex Cloud creates a default security group for all networks, including the Deckhouse Kubernetes Platform cluster network. The default security group contains rules that allow for any traffic to pass in any direction (inbound and outbound) and applies to all subnets within the cloud network, unless an object (VM interface) is explicitly assigned to a different security group. You can change the default security group rules if you need to control traffic in your cluster.

> **✖** Do not delete the default rules that allow for traffic to pass in any direction before finishing configuring all the other rules for the security group. Doing so may disrupt the performance of the cluster.

This section provides general guidelines for setting up a security group. Incorrect configuration of security groups may affect the performance of the cluster. Please consult security group usage details in Yandex Cloud before using it in production environments.

1. Find out in which cloud network the Deckhouse Kubernetes Platform cluster is running.

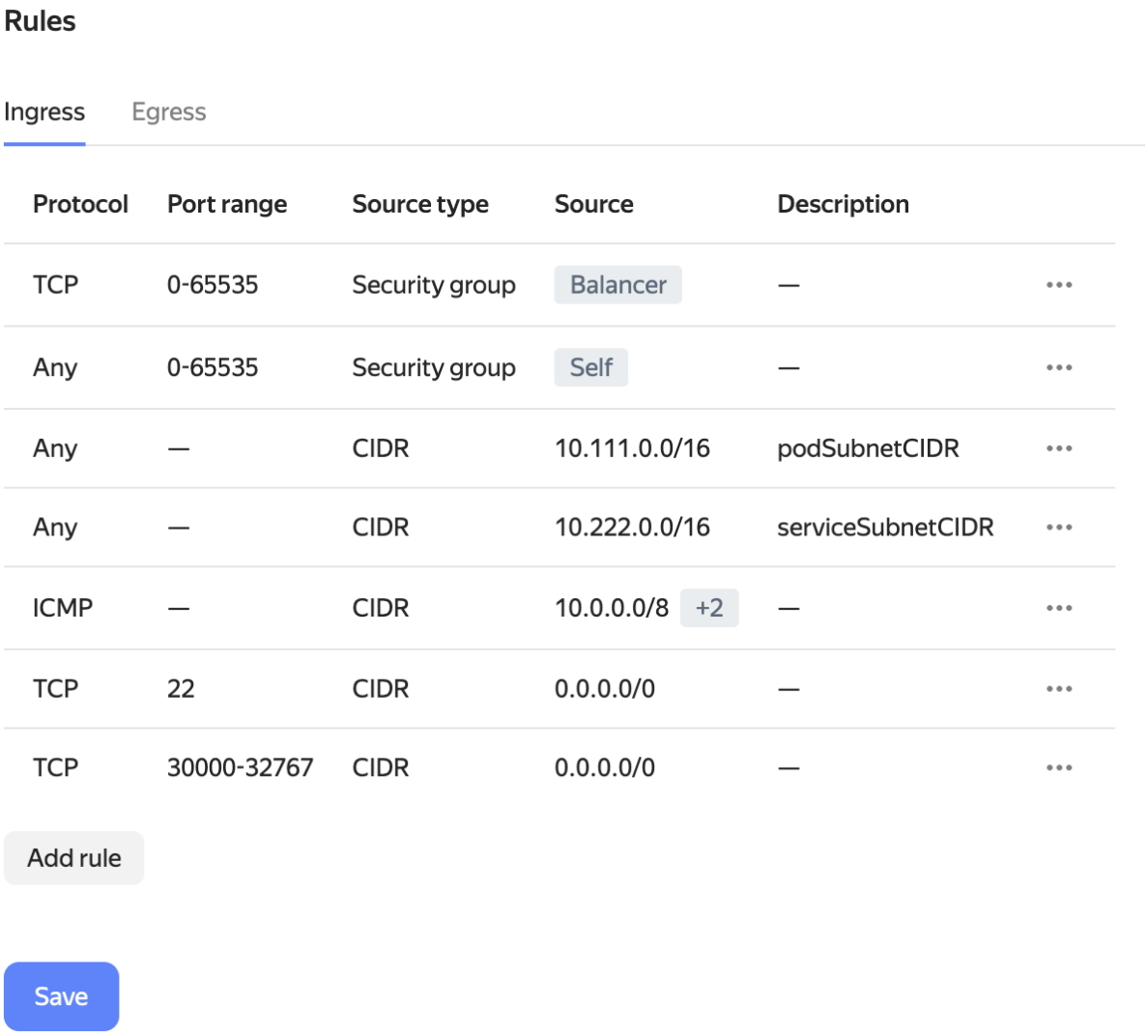
   The network name matches the `prefix` field of the ClusterConfiguration resource. It can be retrieved using the following command:

   ```
   d8 k get secrets -n kube-system d8-cluster-configuration -ojson | \
     jq -r '.data."cluster-configuration.yaml"' | base64 -d | grep prefix | cut -d: -f2
   ```

2. In the Yandex Cloud console, select the Virtual Private Cloud service and navigate to the *Security Groups* section. You should see a single security group labeled `Default`.

3. Create rules as described in Yandex Cloud instructions.



4. Delete the rule that allows for any **inbound** traffic (in the screenshot above it has already been deleted), and save the changes.

Yandex Lockbox integration

The External Secrets Operator allows you to synchronize Yandex Lockbox secrets with the Deckhouse Kubernetes Platform cluster secrets.

The instructions below are meant to be viewed as a *Quick Start* guide. To use integration in production environments, please review the following resources:

- Yandex Lockbox
- Synchronizing with Yandex Lockbox secrets
- External Secrets Operator

Deployment instructions

1. Create a service account required for the External Secrets Operator:

   ```
   yc iam service-account create --name eso-service-account
   ```

2. Create an authorized key for the service account and save it to a file:

   ```
   yc iam key create --service-account-name eso-service-account --output authorized-key.json
   ```

3. Assign `lockbox.editor`, `lockbox.payloadViewer`, and `kms.keys.encrypterDecrypter` roles to a service account to access all catalog secrets:

```
folder_id=<folder id>
yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --
role lockbox.editor
yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --
role lockbox.payloadViewer
yc resource-manager folder add-access-binding --id=${folder_id} --service-account-name eso-service-account --
role kms.keys.encrypterDecrypter
```

For advanced customization, check out access control in Yandex Lockbox.

4. Install the External Secrets Operator using the Helm chart according to instructions.

   Note that you may need to set `nodeSelector`, `tolerations` and other parameters. To do this, use the `./external-secrets/values.yaml` file after unpacking the Helm chart.

   Pull and extract the chart:

   ```
   helm pull oci://cr.yandex/yc-marketplace/yandex-cloud/external-secrets/chart/external-secrets \
     --version 0.5.5 \
     --untar
   ```

   Install the Helm chart:

   ```
   helm install -n external-secrets --create-namespace \
     --set-file auth.json=authorized-key.json \
     external-secrets ./external-secrets/
   ```

   Where:

   - `authorized-key.json` — the name of the file with the authorized key from step 2.

5. Create a SecretStore with the `sa-creds` secret in it:

   ```
   d8 k -n external-secrets apply -f - <<EOF

   apiVersion: external-secrets.io/v1alpha1
   kind: SecretStore
   metadata:
     name: secret-store
   spec:
     provider:
       yandexlockbox:
         auth:
           authorizedKeySecretRef:
             name: sa-creds
             key: key
   EOF
   ```

   Where:

   - `sa-creds` — the name of the `Secret` that contains the authorized key. This secret should show up after the Helm chart has been installed.
   - `key` — the name of the key in the `.data` field of the secret above.

Checking that everything works as expected

1. Check the status of the External Secrets Operator and the secrets store you created:

   ```
   $ d8 k -n external-secrets get po
   NAME                                               READY   STATUS    RESTARTS   AGE
   external-secrets-55f78c44cf-dbf6q                  1/1     Running   0          77m
   external-secrets-cert-controller-78cbc7d9c8-rszhx  1/1     Running   0          77m
   external-secrets-webhook-6d7b66758-s7v9c           1/1     Running   0          77m

   $ d8 k -n external-secrets get secretstores.external-secrets.io
   NAME           AGE   STATUS
   secret-store   69m   Valid
   ```

2. Create a Yandex Lockbox secret with the following parameters:
   - **Name** — `lockbox-secret`.
   - **Key** — enter the non-confidential identifier `password`.
   - **Value** — enter the confidential data to store `p@$$w0rd`.

3. Create an ExternalSecret object that refers to the `lockbox-secret` secret in the `secret-store`:

```
d8 k -n external-secrets apply -f - <<EOF

apiVersion: external-secrets.io/v1alpha1
kind: ExternalSecret
metadata:
  name: external-secret
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: secret-store
    kind: SecretStore
  target:
    name: k8s-secret
  data:
  - secretKey: password
    remoteRef:
      key: <SECRET_ID>
      property: password
EOF
```

Where:

- `spec.target.name` — the name of the new secret. The External Secrets Operator will create this secret in the Deckhouse Kubernetes Platform cluster and populate it with the parameters of the Yandex Lockbox's `lockbox-secret`.
- `spec.data[].secretKey` — the name of the key in the `.data` field of the secret that the External Secrets Operator will create.
- `spec.data[].remoteRef.key` — identifier of the Yandex Lockbox's `lockbox-secret` created earlier, e.g., `e6q28nvfmhu539******`.
- `spec.data[].remoteRef.property` — the **key** you specified earlier for the Yandex Lockbox's `lockbox-secret`.

4. Make sure that the new `k8s-secret` key contains the `lockbox-secret` value:

```
d8 k -n external-secrets get secret k8s-secret -ojson | jq -r '.data.password' | base64 -d
```

The output of the command should contain the **value** of the `password` key of the `lockbox-secret` created earlier:

```
p@$$w0rd
```

Yandex Managed Service for Prometheus integration

This integration lets you use the Yandex Managed Service for Prometheus as an external metrics repository, e.g., for long-term metrics storage.

Writing metrics

1. Create a service account with the `monitoring.editor` role.
2. Create an API key for the service account.
3. Create a PrometheusRemoteWrite resource:

```
d8 k apply -f - <<EOF
apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: yc-remote-write
spec:
  url: <URL_TO_WRITE_METRICS>
  bearerToken: <API_KEY>
EOF
```

Where:

- `<URL TO WRITE_METRICS>` — URL from the Yandex Monitoring/Prometheus/Writing Metrics page.
- `<API KEY>` — the API key you created in the previous step, e.g., `AQVN1HHJRSrfo9jU3aopsXrJyfq_UHs********`.

You may also specify additional parameters; refer to the documentation.

More details about this feature can be found in Yandex Cloud documentation.

Reading metrics with Grafana

1. Create a service account with the `monitoring.viewer` role.
2. Create an API key for the service account.
3. Create a GrafanaAdditionalDatasource resource:

```
d8 k apply -f - <<EOF

apiVersion: deckhouse.io/v1
kind: GrafanaAdditionalDatasource
metadata:
  name: managed-prometheus
spec:
  type: prometheus
  access: Proxy
  url: <URL_READING_METRICS_WITH_GRAFANA>
  basicAuth: false
  jsonData:
    timeInterval: 30s
    httpMethod: POST
    httpHeaderName1: Authorization
  secureJsonData:
    httpHeaderValue1: Bearer <API_KEY>
EOF
```

   Where:

   - `<URL_READING_METRICS_WITH_GRAFANA>` — URL from the Yandex Monitoring/Prometheus/Reading Metrics with Grafana page.
   - `<API_KEY>` — the API key you created in the previous step, e.g., `AQVN1HHJReSrfo9jU3aopsXrJyfq_UHs********`.

   You may also specify additional parameters; refer to the documentation.

More details about this feature can be found in Yandex Cloud documentation.

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

YandexInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of a group of Yandex Instances used by `machine-controller-manager` (the node-manager module).

The `NodeGroup` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalLabels**
    object

    Additional labels.

    Example:

    ```
    additionalLabels:
      project: cms-production
      severity: critical
    ```

  - **spec.additionalSubnets**
    array of strings

    Subnet IDs that VirtualMachines' secondary NICs will connect to. Each subnet listed here translates into one additional network interface.

    Example:

    ```
    additionalSubnets:
    - b0csh41c1or82vuch89v
    - e2lgddi5svochh5fbq96
    ```

  - **spec.assignPublicIPAddress**
    boolean

    Should a public external IPv4 address be assigned to a provisioned Yandex Compute Instance.

    Default: `false`

    Example:

    ```
    assignPublicIPAddress: false
    ```

  - **spec.coreFraction**
    integer

    Percent of reserved CPU capacity on a Yandex Compute Instance. Details…

    Default: `100`

    Allowed values: `5`, `20`, `50`, `100`

    Example:

    ```
    coreFraction: 20
    ```

  - **spec.cores**
    integer

    Required value

Amount of CPU cores to provision on a Yandex Compute Instance.

Example:

```
cores: 4
```

- **spec.diskSizeGB**
integer

  Yandex Compute Instance disk size in gibibytes.

  Default: `50`

  Example:

  ```
  diskSizeGB: 20
  ```

- **spec.diskType**
string

  Instance disk type.

  Size of `network-ssd-nonreplicated` and `network-ssd-io-m3` disks must be a multiple of 93 GB.

  Default: `network-hdd`

  Allowed values: `network-ssd`, `network-hdd`, `network-ssd-nonreplicated`, `network-ssd-io-m3`

  Example:

  ```
  diskType: network-hdd
  ```

- **spec.gpus**
integer

  Number of GPUs on a Yandex Compute Instance.

  Default: `0`

  Example:

  ```
  gpus: 4
  ```

- **spec.imageID**
string

  Image ID to use while provisioning Yandex Compute Instances.

  The masterNodeGroup.instanceClass.imageID parameter will be used by default.

  Example:

  ```
  imageID: fd83ica41cade1mj35sr
  ```

- **spec.mainSubnet**
string

  Subnet ID that VirtualMachines’ primary NIC will connect to.

  If the parameter is not specified, the main network is determined automatically according to the following logic: if a list of networks is set in the existingZoneToSubnetIDMap parameter, then the network is selected from the specified list; otherwise, the created Deckhouse network is used.

  Example:

  ```
  mainSubnet: e9bnc7g9mu9mper9clk4
  ```

- **spec.memory**
integer

  Required value

  Amount of primary memory in MB provision on a Yandex Compute Instance.

  Example:

  ```
  memory: 8192
  ```

- **spec.networkType**

string

Network type: `Standard` or `SoftwareAccelerated`.

> **i** In clusters that use Machine Controller Manager (MCM), changing the `networkType` parameter does not trigger automatic recreation of existing CloudEphemeral nodes. Details are available in the FAQ.

Default: `Standard`

Allowed values: `Standard`, `SoftwareAccelerated`

- **spec.platformID**
  string

  Paltform ID. List of available platforms...

  Default: `standard-v3`

- **spec.preemptible**
  boolean

  Should a provisioned Yandex Compute Instance be preemptible.

  For more information about preemptible virtual machines, read the provider's documentation.

  Default: `false`

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Parameters of a group of Yandex Instances used by `machine-controller-manager` (the node-manager module).

The `NodeGroup` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.additionalLabels**
    object

    Additional labels.

    Example:

    ```
    additionalLabels:
      project: cms-production
      severity: critical
    ```

  - **spec.additionalSubnets**
    array of strings

    Subnet IDs that VirtualMachines' secondary NICs will connect to. Each subnet listed here translates into one additional network interface.

    Example:

    ```
    additionalSubnets:
    - b0csh41c1or82vuch89v
    - e2lgddi5svochh5fbq96
    ```

  - **spec.assignPublicIPAddress**
    boolean

    Should a public external IPv4 address be assigned to a provisioned Yandex Compute Instance.

    Default: `false`

    Example:

    ```
    assignPublicIPAddress: false
    ```

  - **spec.coreFraction**
    integer

Percent of reserved CPU capacity on a Yandex Compute Instance. Details...

Default: `100`

Allowed values: `5`, `20`, `50`, `100`

Example:

```
coreFraction: 20
```

- **spec.cores**
  integer

  Required value

  Amount of CPU cores to provision on a Yandex Compute Instance.

  Example:

  ```
  cores: 4
  ```

- **spec.diskSizeGB**
  integer

  Yandex Compute Instance disk size in gibibytes.

  Default: `20`

  Example:

  ```
  diskSizeGB: 50
  ```

- **spec.diskType**
  string

  Instance disk type.

  Size of `network-ssd-nonreplicated` and `network-ssd-io-m3` disks must be a multiple of 93 GB.

  For more information about possible disk types, read the provider's documentation.

  Default: `network-hdd`

  Allowed values: `network-ssd`, `network-hdd`, `network-ssd-nonreplicated`

  Example:

  ```
  diskType: network-hdd
  ```

- **spec.gpus**
  integer

  Number of GPUs on a Yandex Compute Instance.

  Default: `0`

  Example:

  ```
  gpus: 4
  ```

- **spec.imageID**
  string

  Image ID to use while provisioning Yandex Compute Instances.

  The masterNodeGroup.instanceClass.imageID parameter will be used by default.

  Example:

  ```
  imageID: fd83ica41cade1mj35sr
  ```

- **spec.mainSubnet**
  string

  Subnet ID that VirtualMachines' primary NIC will connect to.

  If the parameter is not specified, the main network is determined automatically according to the following logic: if a list of networks is set in the existingZoneToSubnetIDMap parameter, then the network is selected from the specified list; otherwise, the created Deckhouse network is used.

Example:

```
mainSubnet: e9bnc7g9mu9mper9clk4
```

- **spec.memory**
  integer

  Required value

  Amount of primary memory in MB provision on a Yandex Compute Instance.

  Example:

  ```
  memory: 8192
  ```

- **spec.networkType**
  string

  Network type: STANDARD or SOFTWARE_ACCELERATED

  Default: `STANDARD`

  Allowed values: `STANDARD`, `SOFTWARE_ACCELERATED`

- **spec.platformID**
  string

  Paltform ID. List of available platforms...

  Default: `standard-v2`

- **spec.preemptible**
  boolean

  Should a provisioned Yandex Compute Instance be preemptible.

  For more information about preemptible virtual machines, read the provider's documentation.

  Default: `false`

FAQ

> **i** The module lifecycle stage: General Availability

Setting up the INTERNAL LoadBalancer

Attach the following annotation to the service:

```
yandex.cpi.flant.com/listener-subnet-id: SubnetID
```

The annotation links the LoadBalancer with the appropriate Subnet.

Using a separate target group for a NodeGroup

By default, Yandex Cloud Controller Manager adds all suitable cluster nodes to the default target group. To assign nodes to a separate target group, use the `yandex.cpi.flant.com/target-group-name-prefix` annotation. The annotation is described in the Service annotations section.

1. In the NodeGroup whose nodes should be included in a separate target group, specify the `yandex.cpi.flant.com/target-group-name-prefix` annotation in the `spec.nodeTemplate.annotations` parameter. For example:

   ```
   spec:
     nodeTemplate:
       annotations:
         yandex.cpi.flant.com/target-group-name-prefix: frontend-
   ```

   The annotation is propagated to the Node objects. Yandex Cloud Controller Manager adds all suitable nodes carrying the same annotation value to the target group, regardless of their NodeGroup.

2. In the Service object of the LoadBalancer type, specify the same annotation with the same value. For example:

```
metadata:
  annotations:
    yandex.cpi.flant.com/target-group-name-prefix: frontend-
```

The annotation on the Service identifies the target group used by the LoadBalancer but does not select nodes.

3. Make sure that the values of `yandex.cpi.flant.com/target-group-name-prefix` in the NodeGroup and Service match. In the example above, both resources use the `frontend-` value.
4. The target group will be created with a name generated according to the following pattern:

   ```
   <ANNOTATION_VALUE><YANDEX_CLOUD_CLUSTER_NAME><NETWORK_ID>
   ```

   For example, if the annotation value is `frontend-`, the cluster name is `my-cluster-`, and the network ID is `enp123456789`, the target group name will be:

   ```
   frontend-my-cluster-enp123456789
   ```

Configuration example:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: frontend
spec:
  nodeType: CloudEphemeral
  nodeTemplate:
    annotations:
      yandex.cpi.flant.com/target-group-name-prefix: frontend-
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-frontend
  annotations:
    yandex.cpi.flant.com/target-group-name-prefix: frontend-
spec:
  type: LoadBalancer
```

⚠ Yandex Cloud does not allow a single target, identified by the (`SubnetID, IP`) pair, to belong to multiple target groups at the same time.

When the target group is changed, Yandex Cloud Controller Manager first removes the target from the current group and then adds it to the new one. This transition may cause a brief interruption in traffic handling.

Nodes with a dedicated target group prefix are excluded from the default target group. As a result, load balancers that use the default target group stop routing traffic to those nodes.

This automatic transition is performed only between target groups managed by Yandex Cloud Controller Manager. If a target already belongs to another target group that is not managed by Yandex Cloud Controller Manager, remove it from that group before using the annotation.

After the prefix is changed or removed, the previous target group may remain empty. If it is no longer used by any load balancers, delete it manually.

Reserving a public IP address

This on is used in `externalIPAddresses` and `natInstanceExternalAddress`. It also can be used for a bastion host.

```
$ yc vpc address create --external-ipv4 zone=ru-central1-a
id: e9b4cfmmnc1mhgij75n7
folder_id: b1gog0h9k05lhqe5d88l
created_at: "2020-09-01T09:29:33Z"
external_ipv4_address:
  address: 178.154.226.159
  zone_id: ru-central1-a
  requirements: {}
reserved: true
```

dhcpOptions-related problems and ways to address them

Using DNS servers that differ from those provided by Yandex Cloud in the DHCP settings is a temporary solution. It will be abandoned after Yandex Cloud will introduce the Managed DNS service. To get around the restrictions described below, we recommend using `stubZones` from the `kube-dns` module.

Editing parameters

Pay attention to the following nuances:

1. When changing parameters, you need to invoke `netplan apply` or a similar command that forces the update of the DHCP lease.
2. You will need to restart all hostNetwork Pods (especially `kube-dns`) for the new `resolv.conf` settings to take effect.

Aspects of the use

If the dhcpOptions parameter is set, all DNS are routed to the DNS servers specified. These DNS servers **must** serve DNS requests to the Internet and (if needed) resolve intranet resources.

Do not use this option if the recursive DNSs specified cannot resolve the same list of zones that the recursive DNSs in the Yandex Cloud subnet can resolve.

Changing `networkType` does not recreate CloudEphemeral nodes

In clusters that use Machine Controller Manager (MCM), changing the `networkType` parameter in a YandexInstanceClass resource does not trigger automatic recreation of existing CloudEphemeral nodes. Although the MachineClass is updated, the network acceleration type of existing virtual machines in Yandex Cloud remains unchanged.

DKP intentionally does not take the `networkType` parameter into account when determining whether CloudEphemeral nodes should be recreated in MCM. If it did, updating DKP would recreate CloudEphemeral nodes in all clusters where `networkType` is specified, even if its value had not changed.

In Cluster API, the `networkType` parameter is taken into account when determining whether nodes should be recreated, so changing it automatically triggers node recreation. New CloudEphemeral NodeGroups in Yandex Cloud use Cluster API by default.

If you changed `networkType` in an MCM-based cluster and want the new value to apply to existing CloudEphemeral nodes, recreate them manually — instructions in the documentation.

Setting a custom StorageClass as default

Specify the StorageClass name in the defaultClusterStorageClass parameter in the `global` module settings. Note that after doing so, the `storageclass.kubernetes.io/is-default-class='true'` annotation will be removed from the StorageClass that was previously set as the default one.

```
d8 k edit mc global
```

Resizing a PVC

You can increase the size of an existing PVC by changing the value of the `spec.resources.requests.storage` parameter, without stopping or recreating the pod that uses it.

After the value of `spec.resources.requests.storage` is changed, the CSI driver performs the following operations:

- increases the size of the disk in Yandex Cloud;
- updates the size of the associated PersistentVolume;
- expands the file system on the node to which the volume is attached.

During the operation, the pod continues running, and the mounted volume remains available to the application. After the resize operation is complete, the new file system size becomes available inside the container without restarting the pod.

> **i** Decreasing the size of a PVC is not supported.

To increase the size of a PVC, follow these steps:

1. Get the name of the StorageClass used by the PVC:

   ```
   d8 k -n <NAMESPACE> get pvc <PVC_NAME> \
     -o jsonpath='{.spec.storageClassName}{"\n"}'
   ```

   Where:

   - `<NAMESPACE>` — the namespace containing the PVC;
   - `<PVC_NAME>` — the name of the PVC to resize.

   For example:

   ```
   d8 k -n production get pvc application-data \
     -o jsonpath='{.spec.storageClassName}{"\n"}'
   ```

   Example output:

   ```
   network-ssd
   ```

   In this example, the `application-data` PVC uses the `network-ssd` StorageClass.

2. Make sure that the StorageClass allows volume expansion:

   ```
   d8 k get storageclass <STORAGECLASS_NAME> \
     -o jsonpath='{.allowVolumeExpansion}{"\n"}'
   ```

   Where `<STORAGECLASS_NAME>` is the name of the StorageClass obtained in the previous step.

   For example:

   ```
   d8 k get storageclass network-ssd \
     -o jsonpath='{.allowVolumeExpansion}{"\n"}'
   ```

   Example output:

   ```
   true
   ```

3. Check the current status and size of the PVC:

   ```
   d8 k -n <NAMESPACE> get pvc <PVC_NAME>
   ```

   For example:

   ```
   d8 k -n production get pvc application-data
   ```

   Example output:

   ```
   NAME               STATUS   VOLUME                                     CAPACITY
   ACCESS MODES   STORAGECLASS
   application-data   Bound    pvc-65e92674-077c-4b4f-b65d-19e92f04e103   20Gi       RWO
   network-ssd
   ```

   Make sure that:

   - the PVC is in the `Bound` state;
   - the `CAPACITY` field shows the current PVC size;
   - the `STORAGECLASS` field shows the StorageClass checked in the previous step.

4. Increase the PVC size:

   ```
   d8 k -n <NAMESPACE> edit pvc <PVC_NAME>
   ```

   For example:

```
d8 k -n production edit pvc application-data
```

Specify the new PVC size in the `spec.resources.requests.storage` field:

```
spec:
  resources:
    requests:
      storage: 30Gi
```

In this example, the PVC size is increased to `30Gi`.

Save the changes and close the editor.

> ⚠ For the `network-ssd-nonreplicated` and `network-ssd-io-m3` StorageClasses, the size must be a multiple of 93Gi.

5. Wait for the PVC resize operation to complete:

```
d8 k -n <NAMESPACE> get pvc <PVC_NAME> --watch
```

Where:

- `<NAMESPACE>` — the namespace containing the PVC;
- `<PVC_NAME>` — the name of the PVC being resized.

For example:

```
d8 k -n production get pvc application-data --watch
```

During the resize operation, the `CAPACITY` field may continue to show the previous size:

```
NAME               STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS
application-data   Bound    pvc-65e92674-077c-4b4f-b65d-19e92f04e103   20Gi       RWO
network-ssd
```

The operation is complete when the `CAPACITY` field shows the new PVC size:

```
NAME               STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS
application-data   Bound    pvc-65e92674-077c-4b4f-b65d-19e92f04e103   30Gi       RWO
network-ssd
```

6. Check the PVC events:

```
d8 k -n <NAMESPACE> describe pvc <PVC_NAME>
```

For example:

```
d8 k -n production describe pvc application-data
```

The following events may appear during the resize operation:

```
ExternalExpanding
Resizing
FileSystemResizeRequired
```

The following event indicates that the file system was successfully expanded:

```
FileSystemResizeSuccessful
```

For example:

```
Normal  FileSystemResizeSuccessful  kubelet  MountVolume.NodeExpandVolume succeeded for
volume "pvc-65e92674-077c-4b4f-b65d-19e92f04e103"
```

7. Check the file system size inside the pod:

   ```
   d8 k -n <NAMESPACE> exec <POD_NAME> -- \
     df -hT <MOUNT_PATH>
   ```

   Where:

   - `<NAMESPACE>` — the namespace containing the pod;
   - `<POD NAME>` — the name of the pod using the PVC;
   - `<MOUNT_PATH>` — the path inside the container where the PVC is mounted.

   For example:

   ```
   d8 k -n production exec application-0 -- \
     df -hT /data
   ```

   Example output:

   ```
   Filesystem   Type   Size    Used   Avail   Use%   Mounted on
   /dev/vde     ext4   29.4G   22M    29.4G    1%    /data
   ```

   > **i** The actual file system size may be slightly smaller than the PVC size due to file system metadata.

Adding CloudStatic nodes to a cluster

For VMs that you want to add to the cluster as nodes, add the `node-network-cidr` key to the metadata (Edit VM -> Metadata) with a value equal to the cluster's `nodeNetworkCIDR`.

You can find out the `nodeNetworkCIDR` of the cluster using the command below:

```
d8 k -n kube-system get secret d8-provider-cluster-configuration -o json | jq --raw-output '.data."cloud-
provider-cluster-configuration.yaml"' | base64 -d | grep '^nodeNetworkCIDR'
```

Creating a cluster in a new VPC and setting up a bastion host to access the nodes

1. Bootstrap the base-infrastructure of the cluster:

   ```
   dhctl bootstrap-phase base-infra --config config.yml
   ```

2. Create a bastion host:

   ```
   yc compute instance create \
   --name bastion \
   --hostname bastion \
   --create-boot-disk image-family=ubuntu-2404-lts,image-folder-id=standard-images,size=20,type=network-
   hdd \
   --memory 2 \
   --cores 2 \
   --core-fraction 100 \
   --ssh-key ~/.ssh/<SSH_PUBLIC_KEY_FILE> \
   --zone ru-central1-a \
   --public-address 178.154.226.159
   ```

   > **i** Replace `<SSH_PUBLIC_KEY FILE>` here with the name of your public key. For example, for a key with RSA encryption, it will be `id_rsa.pub`, and for a key with ED25519 encryption, it will be with `id_ed25519.pub`.

3. Continue installing the cluster by specifying the bastion host data. Answer `y` to the question about the Terraform cache:

```
dhctl bootstrap --ssh-bastion-host=178.154.226.159 --ssh-bastion-user=yc-user \
--ssh-user=ubuntu --ssh-agent-private-keys=/tmp/.ssh/<SSH_PRIVATE_KEY_FILE> --config=/config.yml
```

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

Long switching to nodes in lower-priority groups

If switching to nodes in lower-priority groups takes a long time (for example, when node groups with preemptible instances are set to the highest priority and, if such instances are unavailable, switching to nodes from other groups takes a very long time), follow the instructions.

Examples

> **i** The module lifecycle stage: General Availability

Below is an example of the Yandex Cloud cloud provider configuration.

An example of the module configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cloud-provider-yandex
spec:
  version: 1
  enabled: true
  settings:
    additionalExternalNetworkIDs:
    - enp6t4snovl2ko4p15em
```

An example of the `YandexInstanceClass` custom resource

```
apiVersion: deckhouse.io/v1
kind: YandexInstanceClass
metadata:
  name: test
spec:
  cores: 4
  memory: 8192
```

LoadBalancer

Service annotations

Default values are configured in the cluster for placing load balancer resources (the network for the target group and the subnet for the Listener). These values are set automatically during cluster setup and can be overridden with annotations at the individual Service level.

> **i** The default behavior (external or internal LB) depends on the cluster configuration. To explicitly choose the type, use the `yandex.cpi.flant.com/loadbalancer-external` annotation.

The following annotations are supported by Yandex Cloud Controller Manager:

1. `yandex.cpi.flant.com/target-group-network-id` — specifies the NetworkID in which the target group for this Service will be created. Overrides the corresponding default value.
2. `yandex.cpi.flant.com/listener-subnet-id` — sets the SubnetID for the Listeners of the LB created for this Service. Overrides the corresponding default value.
3. `yandex.cpi.flant.com/listener-address-ipv4` — sets a predefined IPv4 address for the Listeners (supported for both internal and external LBs).
4. `yandex.cpi.flant.com/loadbalancer-external` — enables creation of an external LB for this Service (use it when you need to explicitly create an external load balancer). Overrides the default behavior.

5. `yandex.cpi.flant.com/target-group-name-prefix` — specifies the name prefix of the target group used by the LoadBalancer. The annotation on the Service identifies the target group but does not select nodes. To include nodes in the target group, set the same annotation value in `spec.nodeTemplate.annotations` in the NodeGroup. Yandex Cloud Controller Manager includes all suitable nodes carrying this annotation value, regardless of their NodeGroup. The target group name is formed as `<ANNOTATION_VALUE><YANDEX_CLOUD_CLUSTER_NAME><NETWORK_ID>`. For details, see Using a separate target group for a NodeGroup.

If separate target groups are created for the control plane or master nodes, add the label `node.kubernetes.io/exclude-from-external-load-balancers: ""` to the master nodes. This prevents the controller from automatically adding master nodes to new target groups for load balancers.

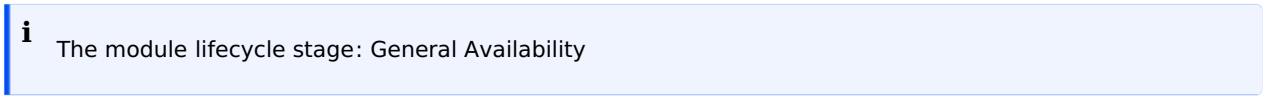
If you create your own load balancer for master nodes and want YCC to also be able to place its load balancers on master nodes, pre-create a target group with a name matching the pattern `${CLUSTER-NAME}${VPC.ID}`.

Target group health checks

Health check parameters (for LB target groups created by the controller):

1. `yandex.cpi.flant.com/healthcheck-interval-seconds` — how often to run the check, in seconds (default: 2).
2. `yandex.cpi.flant.com/healthcheck-timeout-seconds` — how long to wait for an endpoint response, in seconds. If no response is received within this time, the check is considered failed (default: 1).
3. `yandex.cpi.flant.com/healthcheck-unhealthy-threshold` — how many consecutive failed checks are required to mark an endpoint as unhealthy and exclude it from load balancing (default: 2).
4. `yandex.cpi.flant.com/healthcheck-healthy-threshold` — how many consecutive successful checks are required to return an endpoint to healthy status and include it back in load balancing (default: 2).

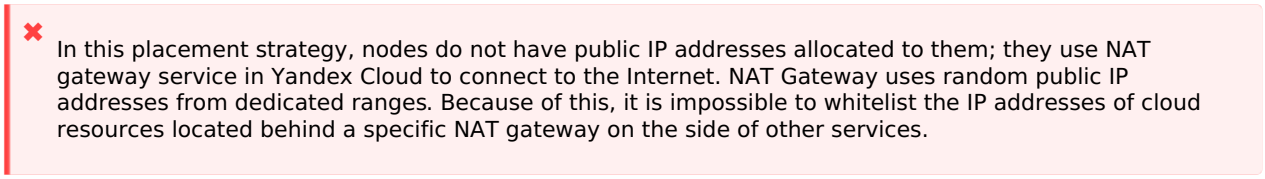
Layouts



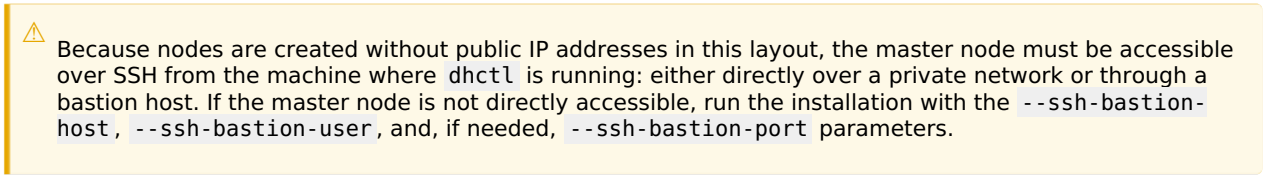
The module lifecycle stage: General Availability

Three layouts are supported. Below is more information about each of them.

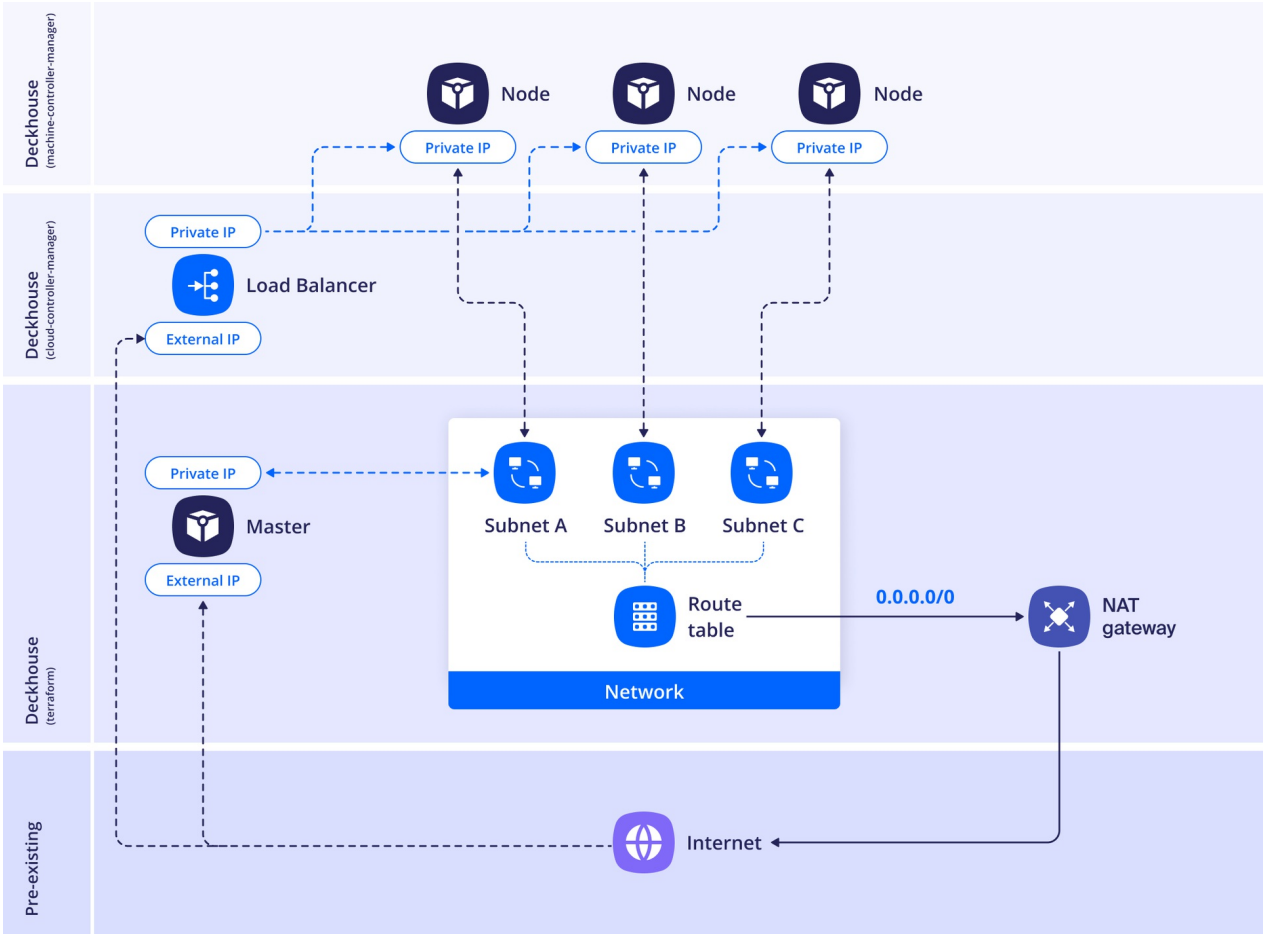
Standard



In this placement strategy, nodes do not have public IP addresses allocated to them; they use NAT gateway service in Yandex Cloud to connect to the Internet. NAT Gateway uses random public IP addresses from dedicated ranges. Because of this, it is impossible to whitelist the IP addresses of cloud resources located behind a specific NAT gateway on the side of other services.



Because nodes are created without public IP addresses in this layout, the master node must be accessible over SSH from the machine where `dhctl` is running: either directly over a private network or through a bastion host. If the master node is not directly accessible, run the installation with the `--ssh-bastion-host`, `--ssh-bastion-user`, and, if needed, `--ssh-bastion-port` parameters.



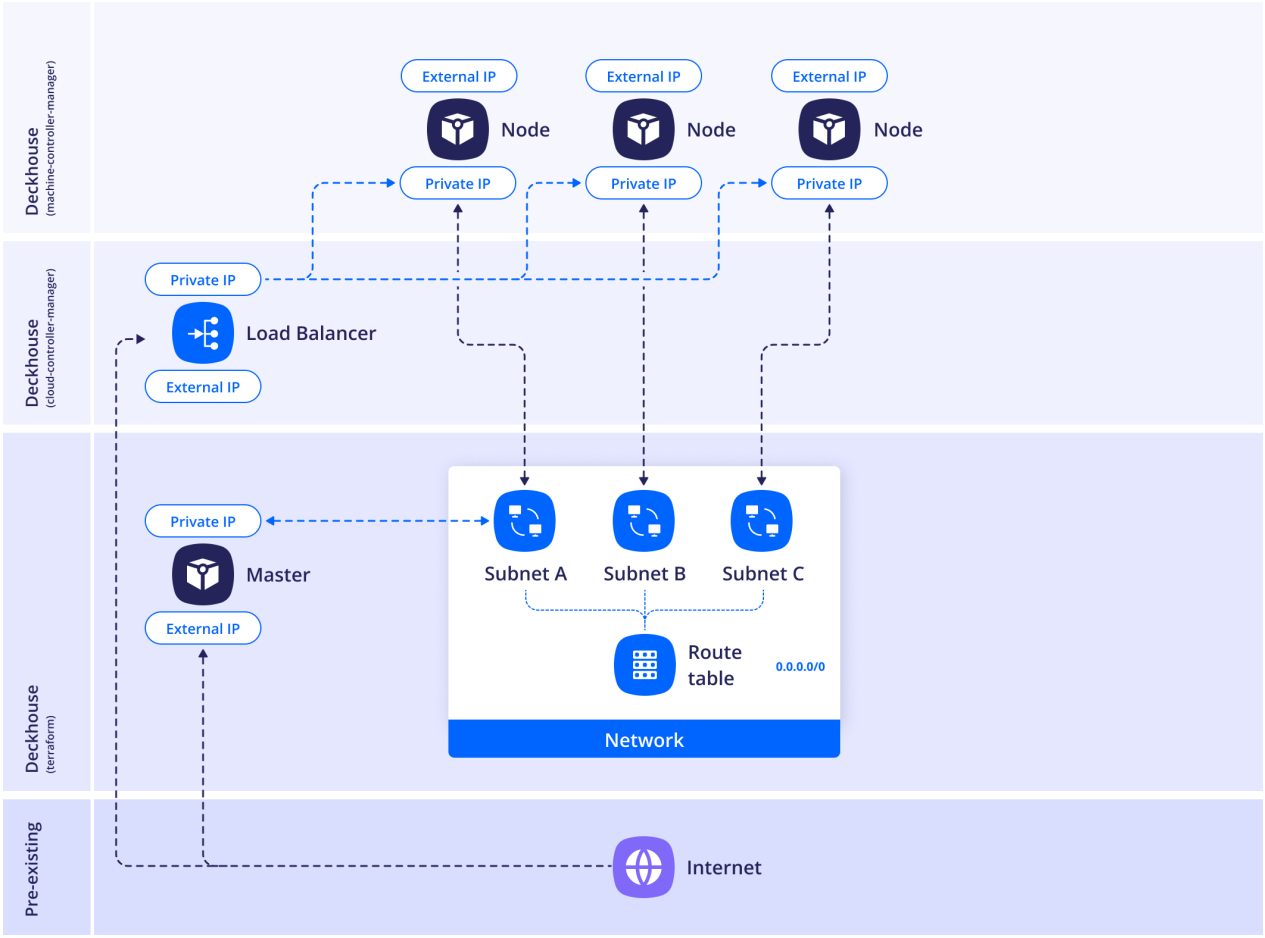
Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: Standard
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 192.168.12.13/24
existingNetworkID: <EXISTING_NETWORK_ID>
provider:
  cloudID: <CLOUD_ID>
  folderID: <FOLDER_ID>
  serviceAccountJSON: |
    {
    "id": "id",
    "service_account_id": "service_account_id",
    "key_algorithm": "RSA_2048",
    "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID....AQAB\n-----END PUBLIC KEY-----\n",
    "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE....1ZPJeBLt+\n-----END PRIVATE KEY-----
\n"
    }
masterNodeGroup:
  replicas: 3
  zones:
  - ru-central1-a
  - ru-central1-b
  - ru-central1-d
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    # Optional: uncomment to assign public IP addresses to master nodes.
    # externalIPAddresses:
    # - "<ZONE_A_EXTERNAL_IP_MASTER_1>"
    # - "Auto"
    # - "Auto"
    # externalSubnetIDs:
    # - <ZONE_A_SUBNET_ID>
    # - <ZONE_B_SUBNET_ID>
    # - <ZONE_D_SUBNET_ID>
    additionalLabels:
      takes: priority
nodeGroups:
- name: worker
  replicas: 2
  zones:
  - ru-central1-a
  - ru-central1-b
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    coreFraction: 50
    # Optional: uncomment to assign public IP addresses to worker nodes.
    # externalIPAddresses:
    # - "Auto"
    # - "Auto"
    # externalSubnetIDs:
    # - <ZONE_A_SUBNET_ID>
    # - <ZONE_B_SUBNET_ID>
    additionalLabels:
      role: example
labels:
  billing: prod
dhcpOptions:
  domainName: test.local
  domainNameServers:
  - <DNS_SERVER_1>
  - <DNS_SERVER_2>
```

WithoutNAT

In this layout, NAT (of any kind) is not used, and each node is assigned a public IP.

> **Caution!** The cloud-provider-yandex module does not support Security Groups, so all cluster nodes will be available without connection restrictions.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: WithoutNAT
provider:
  cloudID: <CLOUD_ID>
  folderID: <FOLDER_ID>
  serviceAccountJSON: |
    {
    "id": "id",
    "service_account_id": "service_account_id",
    "key_algorithm": "RSA_2048",
    "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID....AQAB\n-----END PUBLIC KEY-----\n",
    "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE....1ZPJeBLt+\n-----END PRIVATE KEY-----
\n"
    }
masterNodeGroup:
  replicas: 3
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    externalIPAddresses:
    - "Auto"
    - "Auto"
    - "Auto"
    externalSubnetIDs:
    - <ZONE_A_SUBNET_ID>
    - <ZONE_B_SUBNET_ID>
    - <ZONE_D_SUBNET_ID>
    zones:
    - ru-central1-a
    - ru-central1-b
    - ru-central1-d
nodeGroups:
- name: worker
  replicas: 2
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    coreFraction: 50
    externalIPAddresses:
    - "<ZONE_A_EXTERNAL_IP_WORKER_1>"
    - "Auto"
    externalSubnetIDs:
    - <ZONE_A_SUBNET_ID>
    - <ZONE_B_SUBNET_ID>
    zones:
    - ru-central1-a
    - ru-central1-b
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 192.168.12.13/24
existingNetworkID: <EXISTING_NETWORK_ID>
dhcpOptions:
  domainName: test.local
  domainNameServers:
  - <DNS_SERVER_1>
  - <DNS_SERVER_2>
```

WithNATInstance

In this placement strategy, Deckhouse creates a NAT instance inside new separate subnet and adds a rule to a route table containing a route to `0.0.0.0/0` with a NAT instance as the next hop. The subnet is allocated to prevent routing loops and must not overlap with other networks used in the cluster.

To place the NAT instance in an existing subnet, use the `withNATInstance.internalSubnetID` parameter — the instance will be created in the zone corresponding to that subnet.

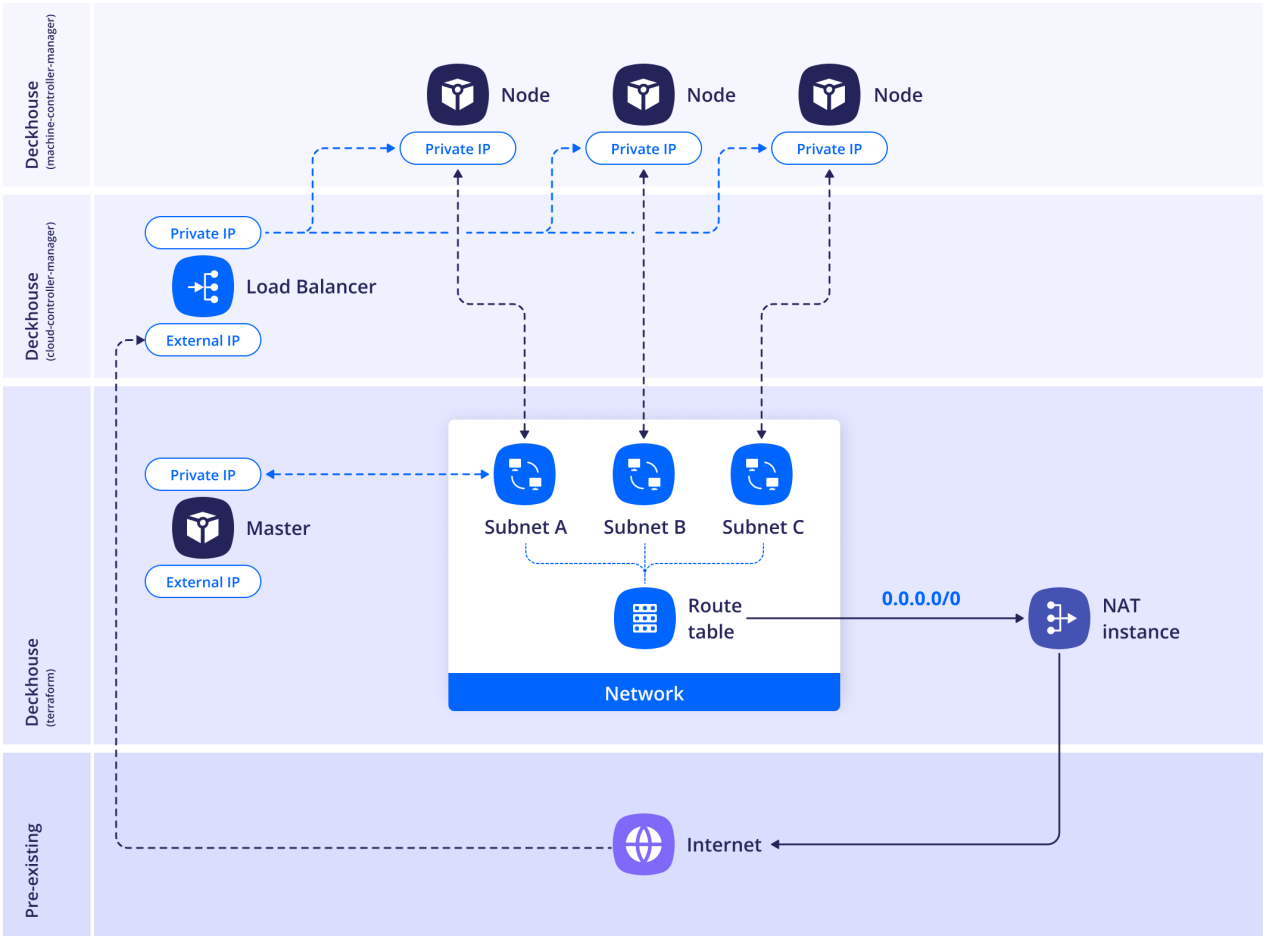
If you need to create a new subnet, specify the `withNATInstance.internalSubnetCIDR` parameter — the NAT instance will be deployed in it.

One of the parameters — `withNATInstance.internalSubnetID` or `withNATInstance.internalSubnetCIDR` — is required.

When the `withNATInstance.internalSubnetCIDR` parameter is specified, the NAT instance will be created in a single instance in zone `a`.

When the `withNATInstance.internalSubnetID` parameter is specified, the NAT instance will be created in a single instance in the zone specified in the designated network settings.

If the `withNATInstance.externalSubnetID` is provided in addition to previous ones, the NAT instance will be attached to it via secondary interface.



Example of the layout configuration:

```
apiVersion: deckhouse.io/v1
kind: YandexClusterConfiguration
layout: WithNATInstance
withNATInstance:
  natInstanceExternalAddress: <NAT_INSTANCE_EXTERNAL_ADDRESS>
  internalSubnetID: <INTERNAL_SUBNET_ID>
  externalSubnetID: <EXTERNAL_SUBNET_ID>
provider:
  cloudID: <CLOUD_ID>
  folderID: <FOLDER_ID>
  serviceAccountJSON: |
    {
    "id": "id",
    "service_account_id": "service_account_id",
    "key_algorithm": "RSA_2048",
    "public_key": "-----BEGIN PUBLIC KEY-----\nMIIwID....AQAB\n-----END PUBLIC KEY-----\n",
    "private_key": "-----BEGIN PRIVATE KEY-----\nMIIE....1ZPJeBLt+\n-----END PRIVATE KEY-----
\n"
    }
masterNodeGroup:
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
nodeGroups:
- name: worker
  replicas: 1
  zones:
  - ru-central1-a
  instanceClass:
    cores: 4
    memory: 8192
    imageID: <IMAGE_ID>
    coreFraction: 50
sshPublicKey: "<SSH_PUBLIC_KEY>"
nodeNetworkCIDR: 192.168.12.13/24
existingNetworkID: <EXISTING_NETWORK_ID>
dhcpOptions:
  domainName: test.local
  domainNameServers:
  - <DNS_SERVER_1>
  - <DNS_SERVER_2>
```

i In this example, master and worker nodes are created without public IP addresses and without additional external network interfaces. Outbound traffic from the cluster subnets goes through the NAT instance.

The NAT instance is not automatically used by the `dhctl` installer as a bastion host or jump host. To install the cluster, the master node must be accessible over SSH from the machine where `dhctl` is running: either directly or through a bastion host.

If the master node does not have a public IP address or is not directly accessible, run the installation with the `--ssh-bastion-host` and `--ssh-bastion-user` parameters.

If you need to connect to the master node directly using a public IP address, specify the `externalIPAddresses` and `externalSubnetIDs` parameters for it. Keep in mind that the interface from the network specified in `externalSubnetIDs` will be used as the node's default gateway.

Cloud provider — zVirt

i The module lifecycle stage: General Availability

The `cloud-provider-zvirt` module integrates Deckhouse Kubernetes Platform with zVirt. It allows the `node-manager` module to use zVirt resources when provisioning nodes for a NodeGroup.

Features of the `cloud-provider-zvirt` module:

- Managing zVirt resources via `cloud-controller-manager`: updates virtual machine and Kubernetes node metadata and removes from Kubernetes nodes that no longer exist in zVirt.
- Provisioning disks via the zVirt CSI driver (`csi.ovirt.org`) so that PersistentVolumes can be requested from the cluster.
- Provisioning CloudPermanent nodes using the Terraform/OpenTofu provider `terraform-provider-ovirt`.
- Provisioning CloudEphemeral nodes via Cluster API (CAPI). Virtual machine parameters are set in the ZvirtInstanceClass resource.
- Registering with `node-manager` so that ZvirtInstanceClass can be used when describing a NodeGroup.
- Enabling CNI for new clusters automatically. By default, `cni-cilium` is used.

Configuration

> **i** The module lifecycle stage: General Availability

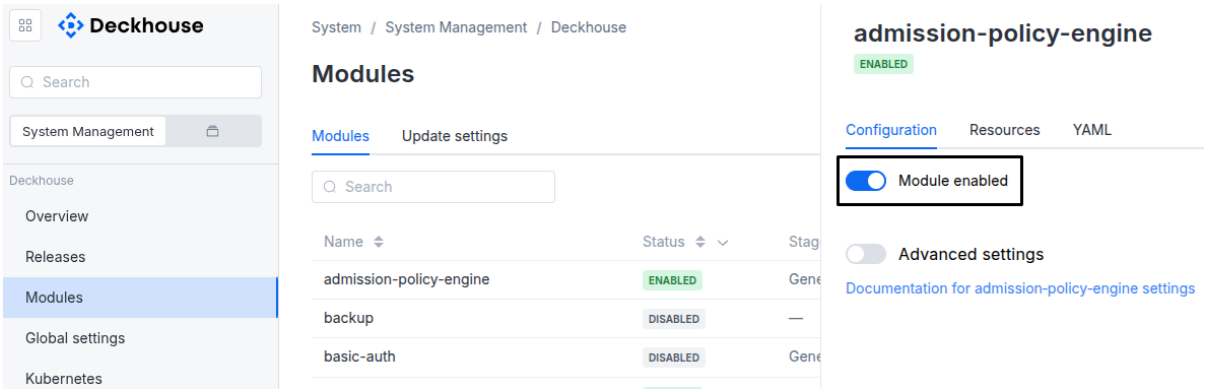
The module is automatically enabled for all cloud clusters deployed in zVirt.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-zvirt` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:

- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable cloud-provider-zvirt
  ```

- **Using ModuleConfig `cloud-provider-zvirt`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cloud-provider-zvirt` (create it if necessary);

  Example of a manifest to enable module `cloud-provider-zvirt`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-zvirt
  spec:
    enabled: true
  ```

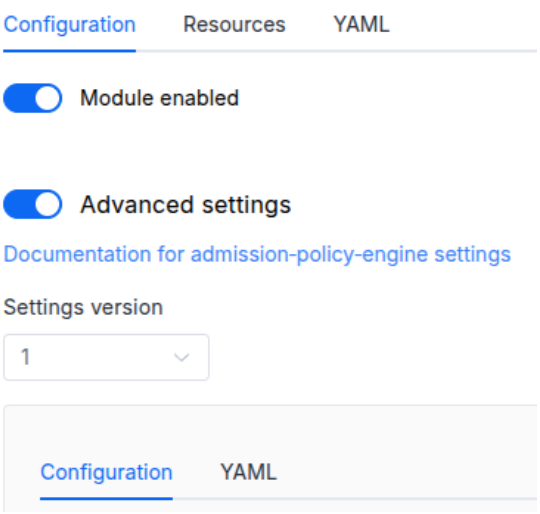
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cloud-provider-zvirt` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

Example:



You can also edit the ModuleConfig object `cloud-provider-zvirt` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cloud-provider-zvirt`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cloud-provider-zvirt` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cloud-provider-zvirt
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cloud-provider-zvirt` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cloud-provider-zvirt`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cloud-provider-zvirt
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

The module does not have any settings.

Provider configuration

> **i** The module lifecycle stage: General Availability

> **i** If the cluster control plane is hosted on a virtual machines or bare-metal servers, the cloud provider uses the settings from the `cloud-provider-zvirt` module in the Deckhouse configuration. Otherwise, if the cluster control plane is hosted in a cloud, the cloud provider uses the ZvirtClusterConfiguration structure for configuration.

ZvirtClusterConfiguration

Version: deckhouse.io/v1

Describes the configuration of a cloud cluster in zVirt.

Used by the cloud provider if a cluster's control plane is hosted in the cloud.

Run the following command to change the configuration in a running cluster:

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

> **i** After updating the node parameters, you need to run the `dhctl converge` command for the changes to take effect. For more information about changing the cluster configuration, see Adding and managing cloud nodes.

Example:

```
apiVersion: deckhouse.io/v1
kind: ZvirtClusterConfiguration
layout: Standard
clusterID: b46372e7-0d52-40c7-9bbf-fda31e187088
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: debian-bookworm
    vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
    storageDomainID: c4bf82a5-b803-40c3-9f6c-b9398378f424
nodeGroups:
- name: worker
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    template: debian-bookworm
    vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
provider:
  server: "<SERVER>"
  username: "<USERNAME>"
  password: "<PASSWORD>"
  insecure: true
```

- **apiVersion**
  string

  Required value

  Allowed values: `deckhouse.io/v1`

- **clusterID**
  string

  Required value

  Cluster ID with shared storage domains and CPUs of the same type to create virtual machines.

  Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{12}$`

  Example:

  ```
  clusterID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
  ```

- **kind**
  string

  Required value

  Allowed values: `ZvirtClusterConfiguration`

- **layout**
  string

  Required value

  The way resources are located in the cloud.

Read more about possible provider layouts.

- **masterNodeGroup**
  object

  Required value

  The definition of the master’s NodeGroup.

  > **i** For the changes to take effect, run `dhctl converge` after modifying the parameters of the `masterNodeGroup` section.

  - **masterNodeGroup.instanceClass**
    object

    Required value

    Partial contents of the fields of the ZvirtInstanceClass.

    - **masterNodeGroup.instanceClass.customNetworkConfig**
      object

      Enables a custom static network configuration for nodes in this NodeGroup.

      If specified, you must set `networkInterfaceName`, `networkInterfaceAddress`, `networkInterfaceGateway` and `networkInterfaceNetmask`.

      - **masterNodeGroup.instanceClass.customNetworkConfig.dnsServers**
        string

        List of DNS servers.

        Example:

        ```
        dnsServers: 8.8.8.8 8.8.4.4
        ```

      - **masterNodeGroup.instanceClass.customNetworkConfig.networkInterfaceAddress**
        array of strings

        List of IP addresses to assign to the interface (one IP address per node).

        Example:

        ```
        networkInterfaceAddress:
        - 192.168.1.10
        - 192.168.1.11
        ```

      - **masterNodeGroup.instanceClass.customNetworkConfig.networkInterfaceGateway**
        string

        Default network gateway.

        Example:

        ```
        networkInterfaceGateway: 192.168.1.1
        ```

      - **masterNodeGroup.instanceClass.customNetworkConfig.networkInterfaceName**
        string

        Name of the network interface to apply the static configuration to.

        Example:

        ```
        networkInterfaceName: enp1s0
        ```

      - **masterNodeGroup.instanceClass.customNetworkConfig.networkInterfaceNetmask**
        string

        Subnet mask for the interface.

        Example:

        ```
        networkInterfaceNetmask: 255.255.255.0
        ```

    - **masterNodeGroup.instanceClass.etcdDiskSizeGb**
      integer

      Etcd disk size in GiB.

Default: `10`

Example:

```
etcdDiskSizeGb: 10
```

- **masterNodeGroup.instanceClass.memory**
integer

  Required value

  Memory in MiB to allocate to the VM.

  Allowed values: `1 <= X`

  Example:

  ```
  memory: 8192
  ```

- **masterNodeGroup.instanceClass.numCPUs**
integer

  Required value

  Number of vCPUs to allocate to the VM.

  Allowed values: `1 <= X`

  Example:

  ```
  numCPUs: 2
  ```

- **masterNodeGroup.instanceClass.rootDiskSizeGb**
integer

  Root disk size in GiB to use in zVirt VirtualMachines.

  Default: `50`

  Example:

  ```
  rootDiskSizeGb: 50
  ```

- **masterNodeGroup.instanceClass.storageDomainID**
string

  Required value

  Storage domain id which contains the shared resources that must be available to all datacenter hosts.

  Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{12}$`

  Example:

  ```
  storageDomainID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
  ```

- **masterNodeGroup.instanceClass.template**
string

  Required value

  Template name to be cloned.

  Example:

  ```
  template: debian-bookworm
  ```

- **masterNodeGroup.instanceClass.vnicProfileID**
string

  Required value

  Virtual NIC profile ID on the basis of which the virtual NIC will be created.

  Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{12}$`

  Example:

```
vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
```

- **masterNodeGroup.replicas**
  integer

  Required value

  The number of master nodes to create. It is important to have an odd number of masters to ensure a quorum.

  Allowed values: `1 <= X`

- **nodeGroups**
  array of objects

  An array of additional NodeGroups for creating static nodes (e.g., for dedicated front nodes or gateways).

  - **nodeGroups.instanceClass**
    object

    Required value

    Partial contents of the fields of the.

    - **nodeGroups.instanceClass.customNetworkConfig**
      object

      Enables a custom static network configuration for nodes in this NodeGroup.

      If specified, you must set `networkInterfaceName`, `networkInterfaceAddress`, `networkInterfaceGateway` and `networkInterfaceNetmask`.

      - **nodeGroups.instanceClass.customNetworkConfig.dnsServers**
        string

        List of DNS servers.

        Example:

        ```
        dnsServers: 8.8.8.8 8.8.4.4
        ```

      - **nodeGroups.instanceClass.customNetworkConfig.networkInterfaceAddress**
        array of strings

        List of IP addresses to assign to the interface (one IP address per node).

        Example:

        ```
        networkInterfaceAddress:
        - 192.168.1.10
        - 192.168.1.11
        ```

      - **nodeGroups.instanceClass.customNetworkConfig.networkInterfaceGateway**
        string

        Default network gateway.

        Example:

        ```
        networkInterfaceGateway: 192.168.1.1
        ```

      - **nodeGroups.instanceClass.customNetworkConfig.networkInterfaceName**
        string

        Name of the network interface to apply the static configuration to.

        Example:

        ```
        networkInterfaceName: enp1s0
        ```

      - **nodeGroups.instanceClass.customNetworkConfig.networkInterfaceNetmask**
        string

        Subnet mask for the interface.

        Example:

        ```
        networkInterfaceNetmask: 255.255.255.0
        ```

- **nodeGroups.instanceClass.etcdDiskSizeGb**
  integer

  Etcd disk size in GiB.

  Default: `10`

  Example:

  ```
  etcdDiskSizeGb: 10
  ```

- **nodeGroups.instanceClass.memory**
  integer

  Required value

  Memory in MiB to allocate to the VM.

  Allowed values: `1 <= X`

  Example:

  ```
  memory: 8192
  ```

- **nodeGroups.instanceClass.numCPUs**
  integer

  Required value

  Number of vCPUs to allocate to the VM.

  Allowed values: `1 <= X`

  Example:

  ```
  numCPUs: 2
  ```

- **nodeGroups.instanceClass.rootDiskSizeGb**
  integer

  Root disk size in GiB to use in zVirt VirtualMachines.

  Default: `50`

  Example:

  ```
  rootDiskSizeGb: 50
  ```

- **nodeGroups.instanceClass.storageDomainID**
  string

  Storage domain id which contains the shared resources that must be available to all datacenter hosts.

  Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{12}$`

  Example:

  ```
  storageDomainID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
  ```

- **nodeGroups.instanceClass.template**
  string

  Required value

  Template name to be cloned.

  Example:

  ```
  template: debian-bookworm
  ```

- **nodeGroups.instanceClass.vnicProfileID**
  string

  Required value

  Virtual NIC profile ID on the basis of which the virtual NIC will be created.

  Pattern: `^[\da-fA-F]{8}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]{4}\-[\da-fA-F]`

`{12}$`

Example:

```
vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
```

- **nodeGroups.name**
  string

  Required value

  The name of the NodeGroup to use for generating node names.

- **nodeGroups.nodeTemplate**

  Parameters of Node objects in Kubernetes to add after registering the node.

  - **nodeGroups.nodeTemplate.annotations**
    object

    The same as the `metadata.annotations` standard field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: 'true'
    ```

  - **nodeGroups.nodeTemplate.labels**
    object

    A list of labels to attach to cluster resources.

    The same as the `metadata.labels` standard field.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **nodeGroups.nodeTemplate.taints**
    array of objects

    The same as the `.spec.taints` field of the Node object.

    > **i** Available fields: `effect`, `key`, and `values`.

    Example:

    ```
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
    ```

    - **nodeGroups.nodeTemplate.taints.effect**
      string

      Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

    - **nodeGroups.nodeTemplate.taints.key**
      string
    - **nodeGroups.nodeTemplate.taints.value**
      string

- **nodeGroups.replicas**
  integer

  Required value

  The number of nodes to create.

- **provider**
  object

  Required value

  Parameters for connecting to the Zvirt.

  - **provider.caBundle**
    string

    CA certificate in base64.

    Default:

  - **provider.insecure**
    boolean

    Set to `true` if Zvirt has a self-signed certificate.

    Default: `false`

  - **provider.password**
    string

    Required value

    The user’s password.

  - **provider.server**
    string

    Required value

    The URL to the Zvirt API endpoint.

  - **provider.username**
    string

    Required value

    The login ID.

- **sshPublicKey**
  string

  Required value

  a public key for accessing nodes.

Preparing environment

> **i** The module lifecycle stage: General Availability

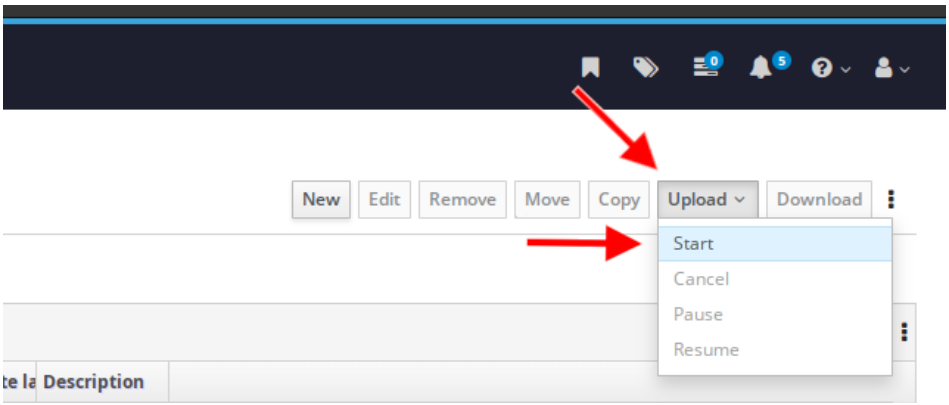
Environment requirements

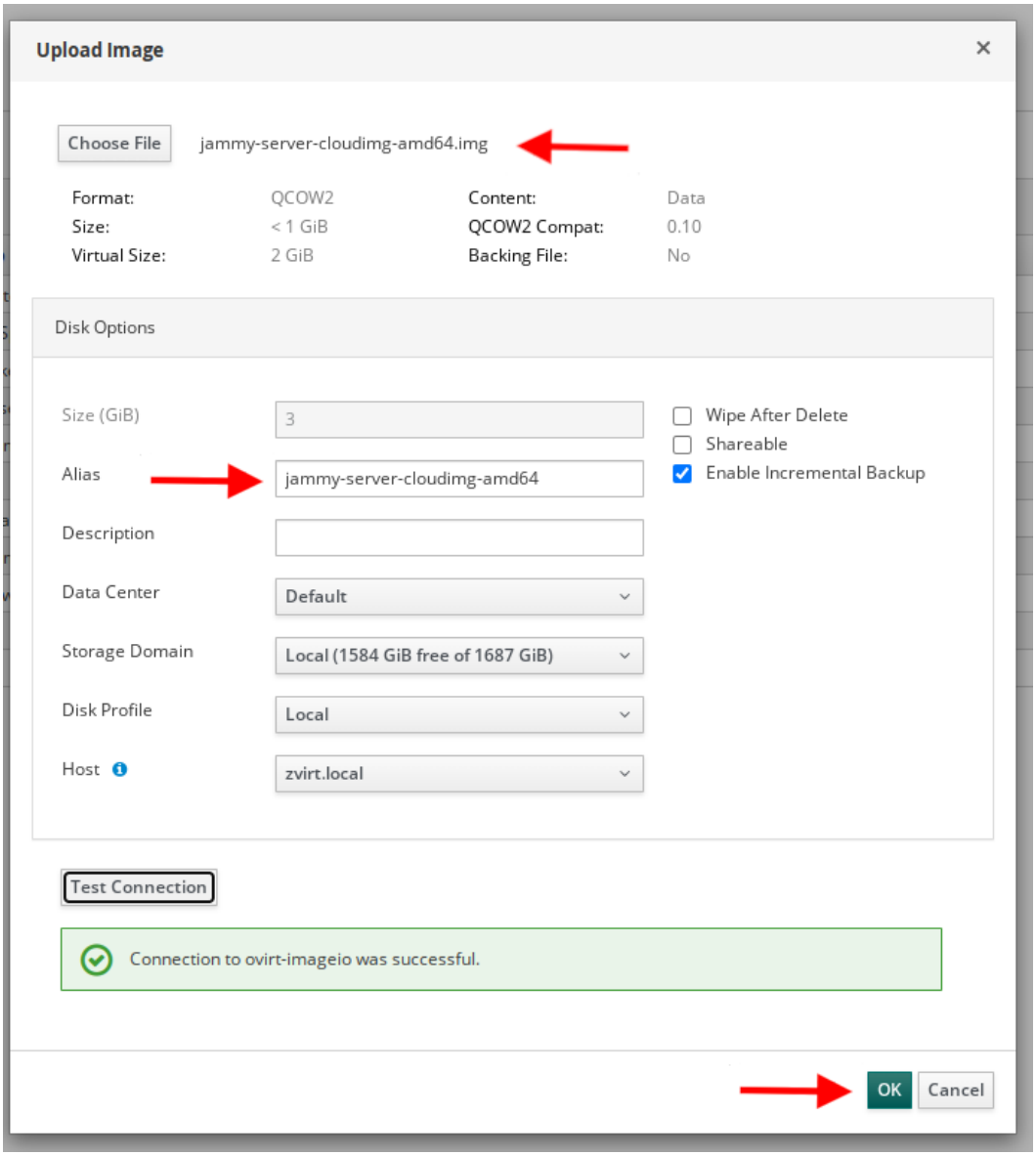
- A working zVirt installation with accessible API.
- zVirt version `4.0-4.4`.
- A user account with permissions to access the API and manage virtual machines.
- A storage domain with an uploaded cloud OS image.
- A prepared virtual machine template based on the cloud image.

Prepare an operating system image

Operating system vendors typically provide special cloud builds of their operating systems for use in virtualization environments. These builds typically contain virtual hardware drivers, cloud-init, virtualization guest agents, and are distributed as IMG or QCOW2 disk images. We recommend that you use these cloud images as the OS on the nodes in your clusters. The cloud image of the operating system must be placed in the zVirt disk storage. Follow these steps to upload the OS image to the storage:

1. Go to the administration portal to the section *zVirt -> Storage -> Disks*.
2. Upload the cloud image of the OS to the repository.





3. Wait until the image has finished loading into the storage. The status “OK” should appear in the “Status” column. This completes the preparation of the OS image.



Prepare a virtual machine template

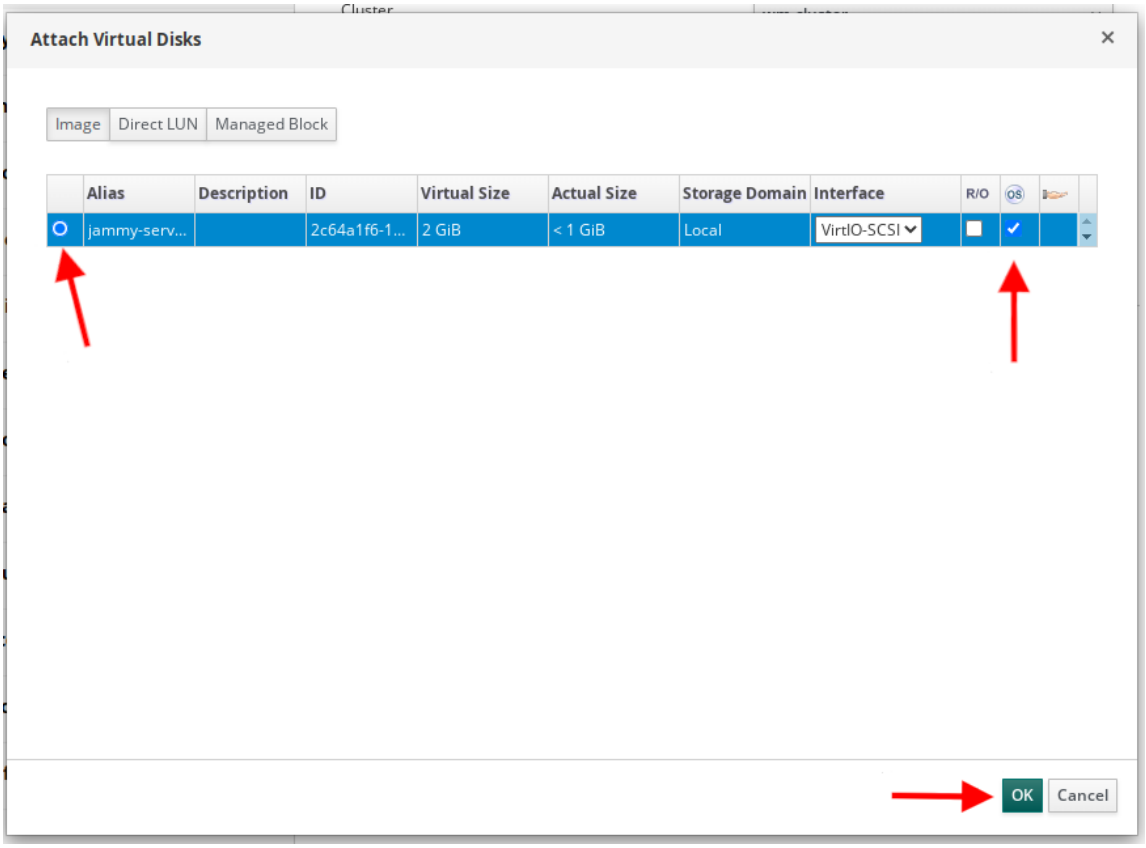
> ⚠ The provider supports working with only one disk in the virtual machine template. Make sure the template contains only one disk.

To create a virtual machine template, go to the *Compute -> Virtual Machines* section of the zVirt Administration Portal and create a new virtual machine with the following parameters:

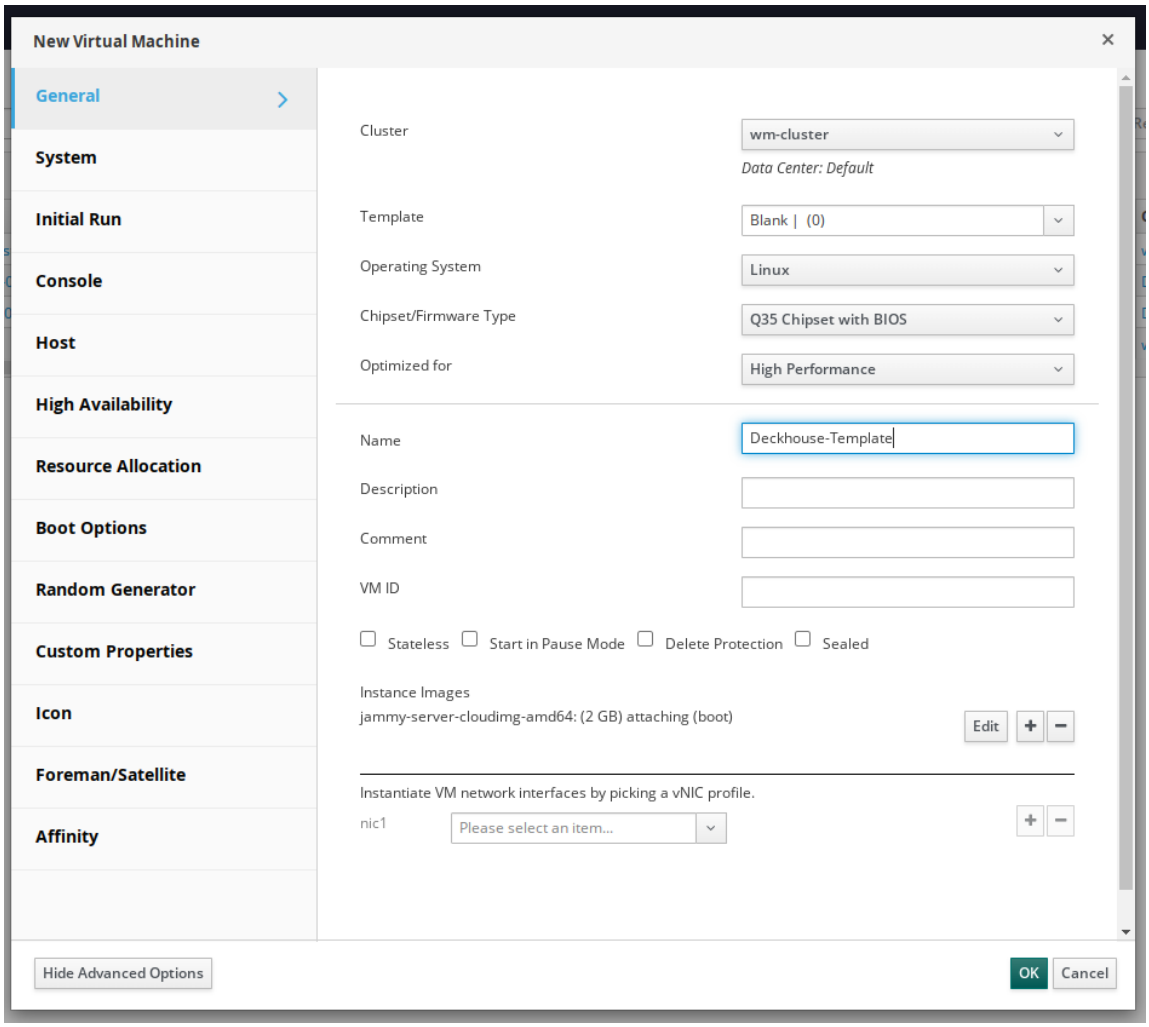
- Section *General*:

- **Template:** `Blank`
- **Operating system:** depending on the OS in the cloud image.
- **Optimized for:** High Performance.

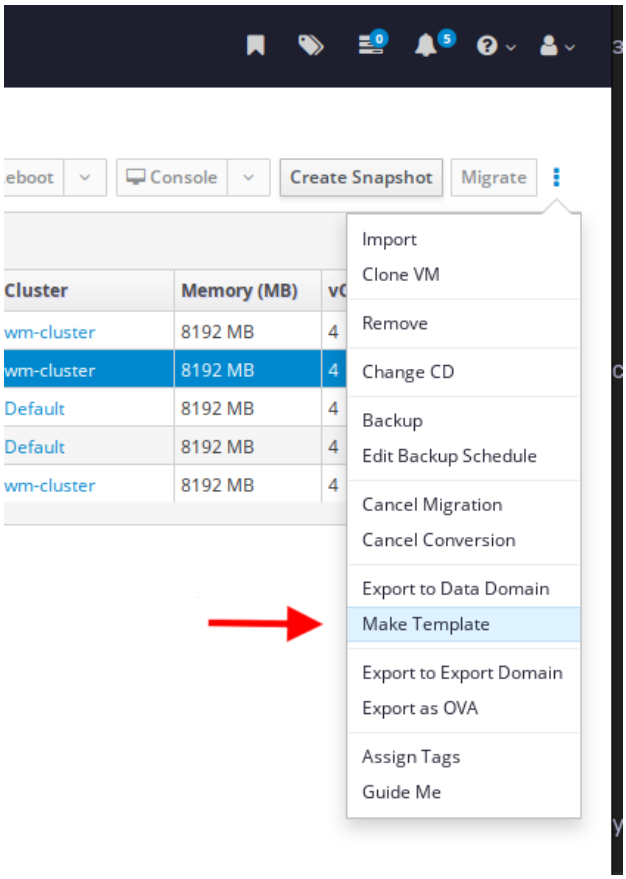
- **Instance Images:** Attach -> Select the previously downloaded cloud-image of the OS, check “Bootable” and select “VirtIO“ interface. No other disks need to be created or attached.



- Network interfaces do not need to be attached, Deckhouse will create and configure them on the virtual machine by itself during installation.

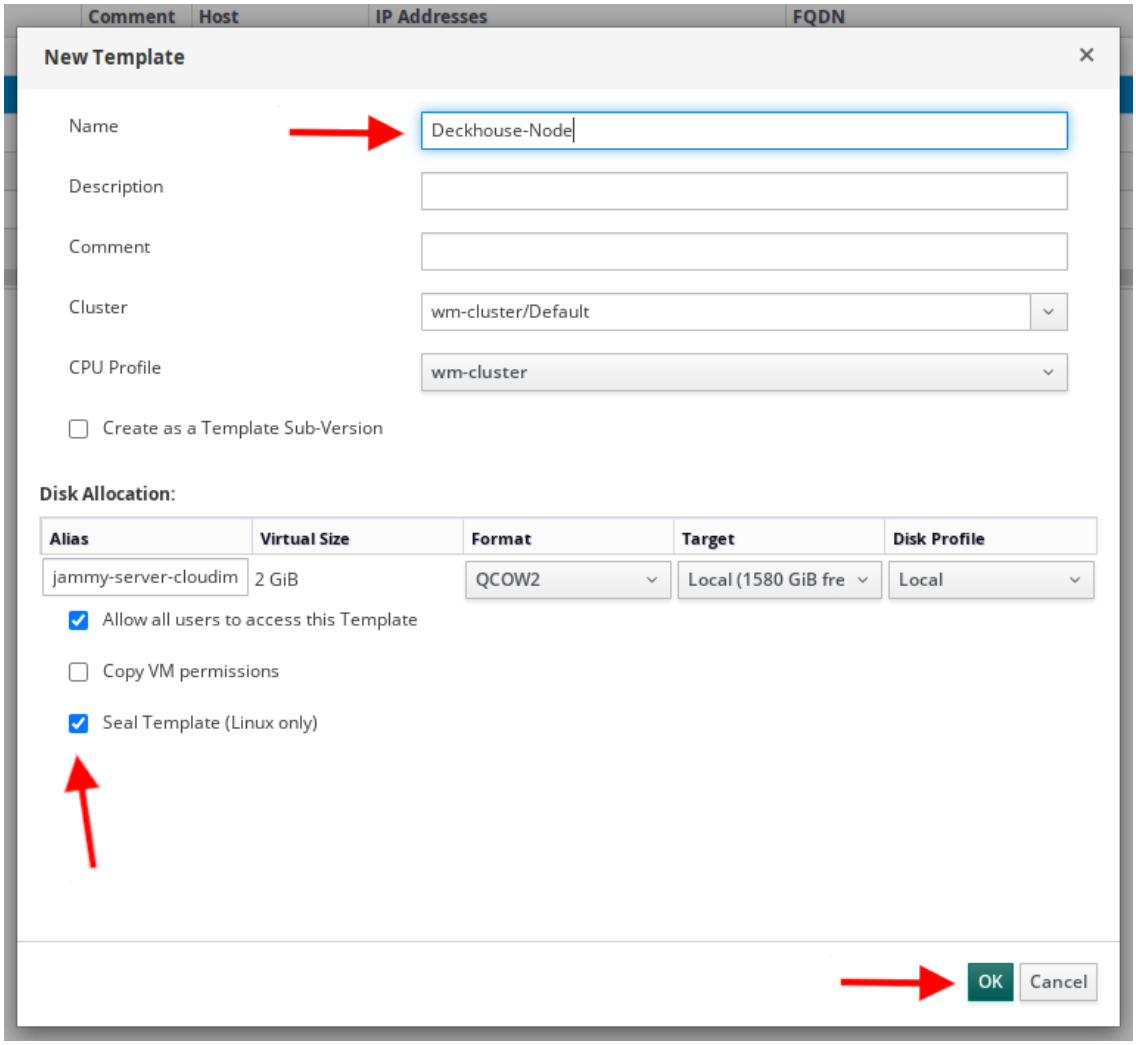


- Leave the rest of the parameters as default and create a virtual machine. When the VM creation process is finished, create a template based on it:
  - Select the created VM in the list and go to the window of creating a new template.



  - Fill in the template parameters, only the name is mandatory. Enable the parameter *Seal Template*

(Linux only):



While the template is being created, the virtual machine and the cloud image disk will go into a state of *Image locked*. You can check that the template was successfully created in the section *Compute -> Templates*:

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

ZvirtInstanceClass

Scope: Cluster

- v1
- v1alpha1

Parameters of a group of zVirt VirtualMachines used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.memory**
    integer

Required value

Memory in MiB to allocate to zVirt VirtualMachines.

Example:

```
memory: 8192
```

- **spec.numCPUs**
  integer

  Required value

  Count of vCPUs to allocate to zVirt VirtualMachines.

  Example:

  ```
  numCPUs: 2
  ```

- **spec.rootDiskSizeGb**
  integer

  Root disk size in GiB to use in zVirt VirtualMachines.

  The disk will be automatically resized if its size in the template differs from specified.

  Default: `30`

  Example:

  ```
  rootDiskSizeGb: 30
  ```

- **spec.template**
  string

  Required value

  Template name to be cloned.

  Example:

  ```
  template: debian-bookworm
  ```

- **spec.vnicProfileID**
  string

  Required value

  Virtual NIC profile ID on the basis of which the virtual NIC will be created.

  Pattern: `^[0-9a-fA-F]{8}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{12}$`

  Example:

  ```
  vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

Parameters of a group of zVirt VirtualMachines used by `machine-controller-manager` (the node-manager module).

The `CloudInstanceClass` resource of the `node-manager` module refers to this resource.

- **spec**
  object

  Required value

  - **spec.memory**
    integer

    Required value

    Memory in MiB to allocate to zVirt VirtualMachines.

    Example:

```
memory: 8192
```

- **spec.numCPUs**
  integer

  Required value

  Count of vCPUs to allocate to zVirt VirtualMachines.

  Example:

  ```
  numCPUs: 2
  ```

- **spec.rootDiskSizeGb**
  integer

  Root disk size in GiB to use in zVirt VirtualMachines.

  The disk will be automatically resized if its size in the template differs from specified.

  Default: `30`

  Example:

  ```
  rootDiskSizeGb: 30
  ```

- **spec.template**
  string

  Required value

  Template name to be cloned.

  Example:

  ```
  template: debian-bookworm
  ```

- **spec.vnicProfileID**
  string

  Required value

  Virtual NIC profile ID on the basis of which the virtual NIC will be created.

  Pattern: `^[0-9a-fA-F]{8}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{12}$`

  Example:

  ```
  vnicProfileID: 49bb4594-0cd4-4eb7-8288-8594eafd5a86
  ```

- **status**
  object
  - **status.nodeGroupConsumers**
    array of strings

FAQ

> **i** The module lifecycle stage: General Availability

How to get vNicProfileId

VNicProfileId can be obtained by querying the zVirt API

```
curl -u "<user>@<profile>:<password>" -X GET https://<zVirt API URL>/vnicprofiles
```

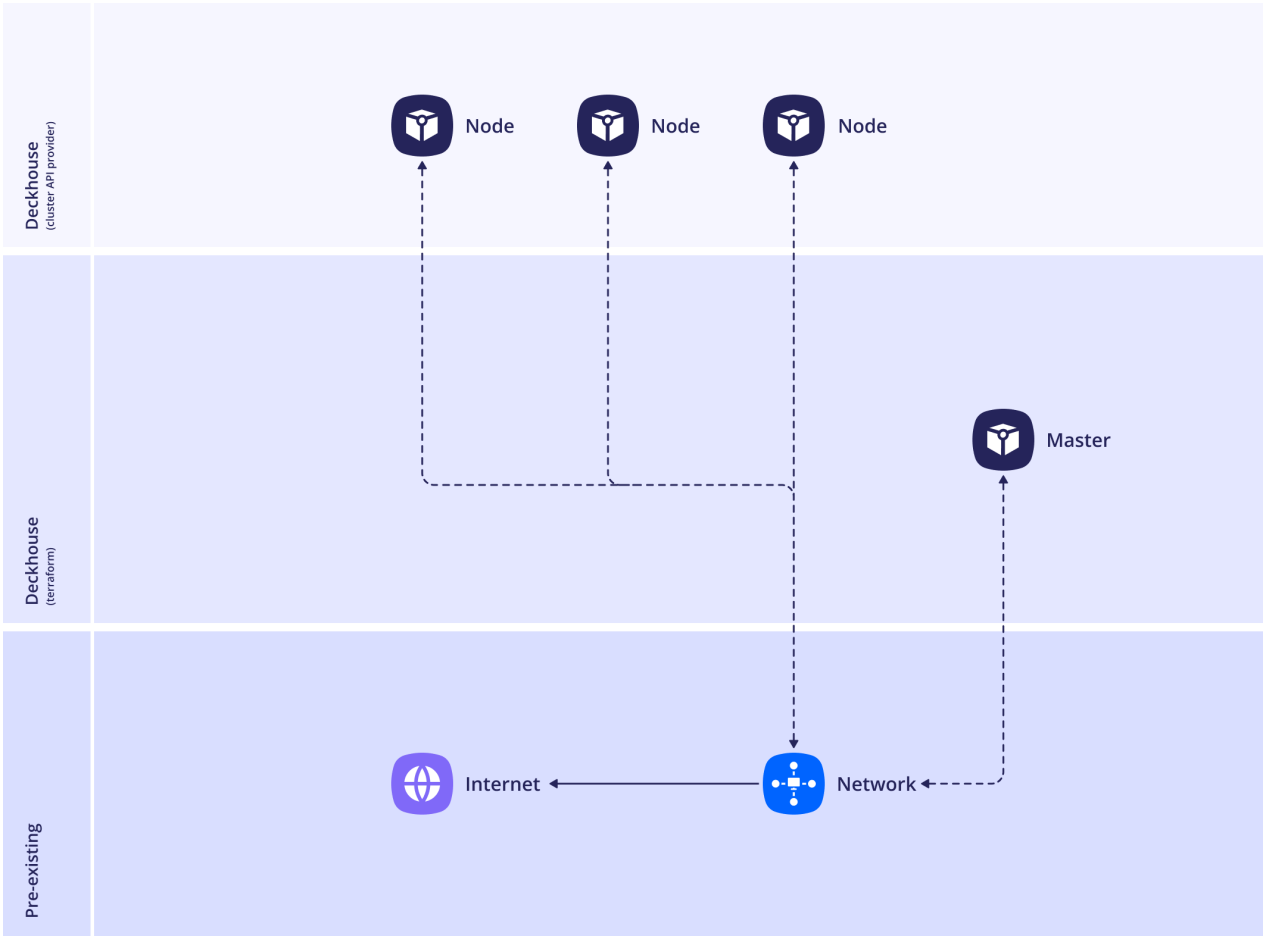
Response example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<vnic_profiles>
   <vnic_profile href="/ovirt-engine/api/vnicprofiles/49bb4594-0cd4-4eb7-8288-8594eafd5a86"
id="49bb4594-0cd4-4eb7-8288-8594eafd5a86">
      <name>vm-net-01</name>
      <link href="/ovirt-engine/api/vnicprofiles/49bb4594-0cd4-4eb7-8288-
8594eafd5a86/permissions" rel="permissions"/>
      <pass_through>
         <mode>disabled</mode>
      </pass_through>
      <port_mirroring>false</port_mirroring>
      <network href="/ovirt-engine/api/networks/74a741c9-0d40-4008-8e58-1c903ee6eba7"
id="74a741c9-0d40-4008-8e58-1c903ee6eba7"/>
   </vnic_profile>
   ...
</vnic_profiles>
```

Cloud provider - zVirt: Layouts

> **i** The module lifecycle stage: General Availability

Standard



Example of the layout configuration:

```
---
apiVersion: deckhouse.io/v1
kind: ZvirtClusterConfiguration
layout: Standard
sshPublicKey: "ssh-rsa AAAABBBB"
clusterID: "b46372e7-0d52-40c7-9bbf-fda31e187088"
provider:
  server: '<SERVER>'
  username: '<USERNAME>'
  password: '<PASSWORD>'
  insecure: true
masterNodeGroup:
  replicas: 1
  instanceClass:
    numCPUs: 4
    memory: 8192
    rootDiskSizeGb: 40
    template: ALT-p10
    vnicProfileID: "49bb4594-0cd4-4eb7-8288-8594eafd5a86"
    storageDomainID: "c4bf82a5-b803-40c3-9f6c-b9398378f424"
```

The cni-cilium module

> **i** Available with limitations in: **CE, BE, SE**
>
> Available without limitations in: **SE+, EE**

> **i** The module lifecycle stage: General Availability

The `cni-cilium` module provides a network in a cluster. It is based on the Cilium project.

Limitations

1. Services with type `NodePort` and `LoadBalancer` are incompatible with hostNetwork endpoints in LB mode `DSR`. Switch to `SNAT` mode if it is required.
2. `HostPort` pods only bind to one IP address. If the OS has multiple interfaces/IP, Cilium will choose one, preferring `private` to `public`.
3. To ensure the stable operation of `cni-cilium` on cluster nodes, disable Elastic Agent or restrict its access to the Elastic management server. Elastic Agent includes an Elastic Endpoint component which uses the Extended Berkeley Packet Filter (eBPF) technology on cluster nodes and may remove critical eBPF programs required for `cni-cilium` to work correctly. For detailed information and discussion of the issue, refer to the publications of the Cilium and Elastic projects.
4. Kernel requirements:
   - Linux kernel version not lower than `5.8` for the `cni-cilium` module to work and work together with the istio, openvpn or node-local-dns modules.
5. OS compatibility:
   - Ubuntu:
     - incompatible with version 18.04;
     - HWE kernel installation required for working with version 20.04.
   - Astra Linux:
     - incompatible with the “Smolensk” edition.
   - CentOS:
     - for versions 7 and 8, a new kernel from the repository is required.

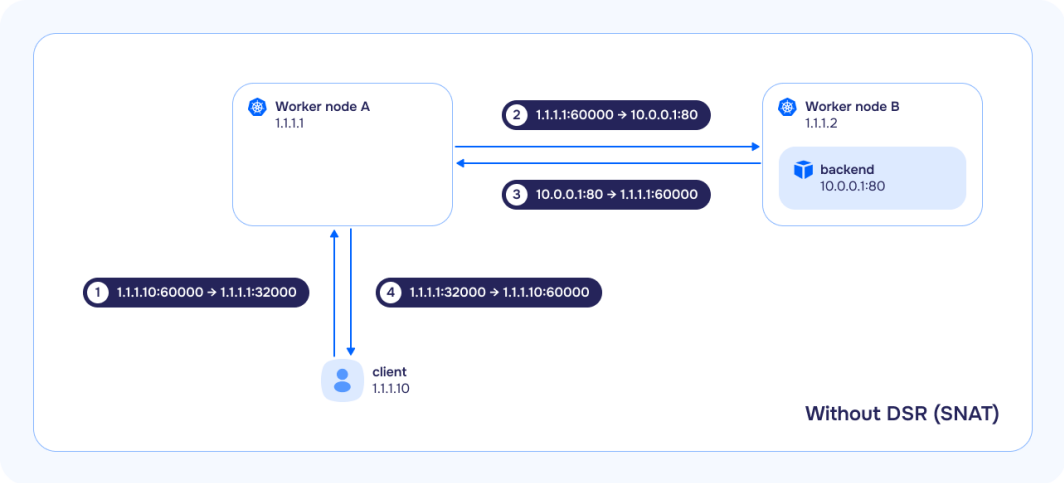
Handling external traffic in different `bpfLB` modes (replacing kube-proxy from Cilium)

Kubernetes typically uses schemes where traffic comes to a balancer that distributes it among many servers. Both incoming and outgoing traffic passes through the balancer. Thus, the total throughput is limited by the resources and channel width of the balancer. To optimize traffic and unload the balancer, the `DSR` mechanism was invented, in which incoming packets go through the balancer, and outgoing ones go directly from the terminating servers. Since responses are usually much larger in size than requests, this approach can significantly increase the overall throughput of the scheme.

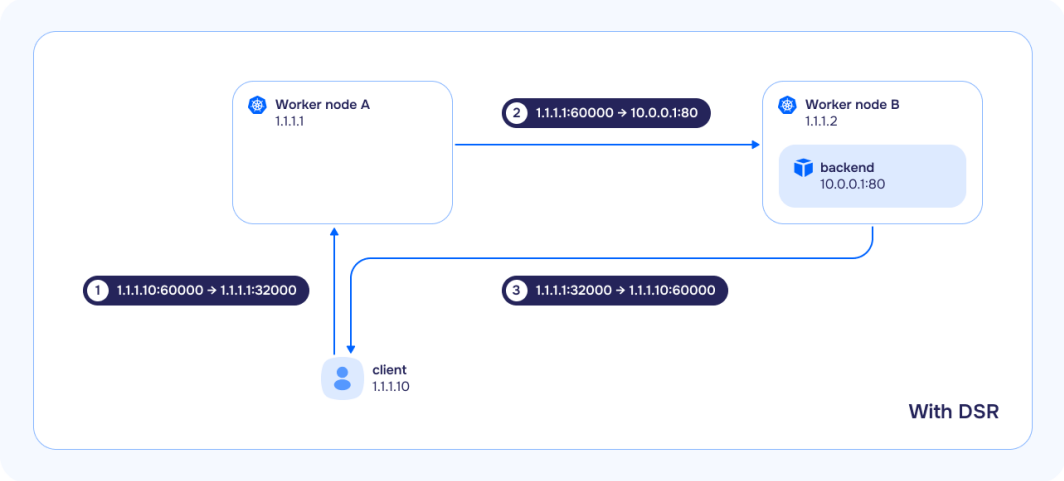
To extend the capabilities, the module allows selectable mode of operation, which affects the behavior of `Service` with the `NodePort` and `LoadBalancer` types:

- `SNAT` (Source Network Address Translation) — is a subtype of NAT in which, for each outgoing packet, the source IP address is translated to the IP address of the gateway from the target subnet, and incoming

packets passing through the gateway are translated back based on a translation table. In this mode, `bpfLB` fully replicates the logic of `kube-proxy`:
  - if `externalTrafficPolicy: Local` is specified in the `Service`, the traffic will be forwarded and balanced only to those target pods running on the same node where the traffic arrived. If the target pod is not running on this node, the traffic will be dropped.
  - if `externalTrafficPolicy: Cluster` is specified in the `Service`, the traffic will be forwarded and balanced to all target pods in the cluster. At the same time, if the target pods are located on other nodes, SNAT will be performed when transmitting traffic to them (the source IP address will be replaced with the InternalIP of the node).



- `DSR` (Direct Server Return) — is a method where all incoming traffic passes through the load balancer, and all outgoing traffic bypasses it. This method is used instead of `SNAT`. Often, responses are much larger than requests, and `DSR` can significantly increase the overall throughput of the scheme:
  - if `externalTrafficPolicy: Local` is specified in the `Service`, its behavior is completely analogous to `kube-proxy` and `bpfLB` in `SNAT` mode.
  - if `externalTrafficPolicy: Cluster` is specified in the `Service`, the traffic will be forwarded and balanced to all target pods in the cluster. It is important to take into account the following features:
    - if the target pods are on other nodes, then the source IP address will be preserved when incoming traffic is sent to them;
    - outgoing traffic will go directly from the node on which the target pod was launched;
    - the source IP address will be replaced with the external IP address of the node to which the incoming request **originally** came.



⚠ In case of using `DSR` and `Service` mode with `externalTrafficPolicy: Cluster` additional network environment settings are required. Network equipment must be ready for asymmetric traffic flow: IP address anti-spoofing tools (`uRPF`, `sourceGuard`, etc.) must be disabled or configured accordingly.

- `Hybrid` — in this mode, TCP traffic is processed in `DSR` mode, and UDP in `SNAT` mode.

Using CiliumClusterwideNetworkPolicies

✖ Using CiliumClusterwideNetworkPolicies if the `policyAuditMode` option is absent in the cni-cilium module settings may lead to incorrect operation of Control plane or loss of SSH access to all cluster nodes.

Follow these steps to use CiliumClusterwideNetworkPolicies:

1. Apply the primary set of `CiliumClusterwideNetworkPolicy` objects. To do this, in the settings of the cni-cilium module add the configuration option `policyAuditMode` with the value `true`. The option can be removed after applying all `CiliumClusterwideNetworkPolicy` objects and verifying their functionality in Hubble UI.
2. Apply network security policy rule:

```
apiVersion: "cilium.io/v2"
kind: CiliumClusterwideNetworkPolicy
metadata:
  name: "allow-control-plane-connectivity"
spec:
  ingress:
  - fromEntities:
    - kube-apiserver
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/control-plane: ""
```

If CiliumClusterwideNetworkPolicies are not used, the control plane may work incorrectly for up to a minute during the reboot of `cilium-agent` pods. This occurs due to Conntrack table reset. Binding to the `kube-apiserver` entity helps to bypass the bug.

Changing Cilium Operation Mode

When changing Cilium's operation mode (the tunnelMode parameter) from `Disabled` to `VXLAN` or vice versa, it is necessary to reboot all nodes, otherwise, pod availability issues may occur.

Disabling the kube-proxy Module

Cilium fully replaces the functionality of the `kube-proxy` module, so `kube-proxy` is automatically disabled when the `cni-cilium` module is enabled.

Using selective load balancing algorithm for services

In Deckhouse Kubernetes Platform, you can apply the following algorithms to load balance service traffic:

- `Random`: Randomly select a backend for each connection. Easy to implement, but does not always provide even distribution.
- `Maglev`: Uses consistent hashing to distribute traffic evenly, suitable for large-scale services.
- `Least Connections`: Directs traffic to the backend with the lowest number of active connections, optimizing load for applications with long-lived connections.

By default, the **Random** balancing algorithm is set for all services. However, Deckhouse allows you to override the algorithm for individual services. To use a selective balancing algorithm for a specific service, follow these steps:

- Edit the `cni-cilium` module configuration in Deckhouse by enabling the `extraLoadBalancerAlgorithmsEnabled` parameter. This activates support for service annotations for selective algorithms.
- In the service manifest, specify the `service.cilium.io/lb-algorithm` annotation with one of the values: `random`, `maglev`, or `least-conn`.

⚠ This mechanism requires Linux kernel version 5.15 or higher to work correctly.

Using Egress Gateway

Egress Gateway in Deckhouse Kubernetes Platform can be used in one of two modes: Basic and Mode with Virtual IP. Use Custom Resource EgressGateway (parameter `spec.sourceIP.node`) to select the mode.

Basic mode

Pre-configured IP addresses are used on egress nodes.

Virtual IP mode

The ability to dynamically assign additional IP addresses to nodes is implemented.

Exporting data from Hubble

Deckhouse Kubernetes Platform allows to configure data export from Hubble running inside Cilium agents using the cluster-scoped custom resource `HubbleMonitoringConfig`. To enable export, create a HubbleMonitoringConfig resource.

⚠ Creating or modifying the HubbleMonitoringConfig resource will **restart all Cilium agents** in the cluster.

Third-party components

List of third-party software used in the `cni-cilium` module:

- Cilium 1.17.4

  License: Apache License 2.0

  Cilium is open source software for providing and transparently securing network connectivity and loadbalancing between application workloads such as application containers or processes.

- CNI plugins 1.7.1

  License: Apache License 2.0

  Some reference and example networking plugins, maintained by the CNI team.

- Gops 0.3.27

  License: BSD 3-Clause License

  A tool to list and diagnose Go processes currently running on your system.

Configuration

ℹ Available with limitations in: **CE, BE, SE**

Available without limitations in: **SE+, EE**

ℹ The module lifecycle stage: General Availability

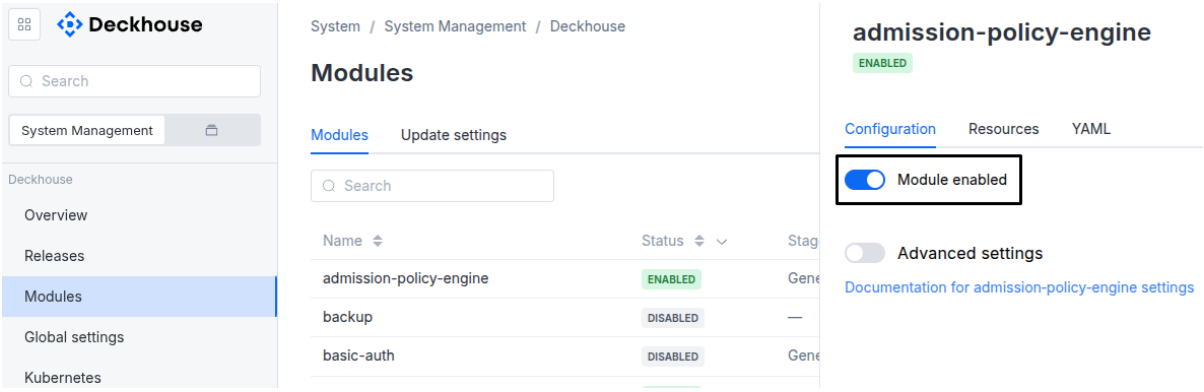
The module has 8 alerts.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cni-cilium` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

Example of enabling the module:

```
d8 system module enable cni-cilium
```

- **Using ModuleConfig `cni-cilium`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cni-cilium` (create it if necessary);

  Example of a manifest to enable module `cni-cilium`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  enabled: true
```

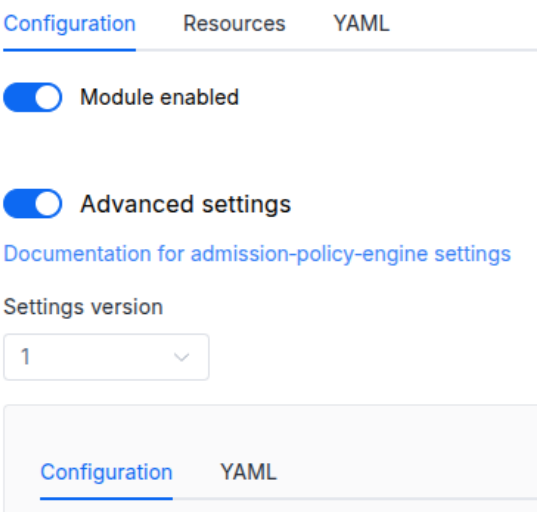
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cni-cilium` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `cni-cilium` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cni-cilium`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cni-cilium` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc cni-cilium
```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cni-cilium` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

  Example of a manifest for ModuleConfig `cni-cilium`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-cilium
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.activeL2NeighborDiscoveryEnabled**
    boolean

    Enables the Cilium `L2 neighbor discovery` mechanism, which helps ensure that ARP table entries are kept up-to-date on all cluster nodes.

    Default: `true`
  - **settings.bpfLBMode**
    string

    eBPF LoadBalancer mode:

    - `SNAT` — traffic from the client to the pod (and back) passes through NAT, and accordingly the sender's address is lost.
    - `DSR` — traffic from the client to the pod passes with the sender's address preserved, and back - according to the routing rules (bypassing the balancer). This mode saves network traffic and reduces delays, but only works for TCP traffic.
    - `Hybrid` — TCP traffic is processed in DSR mode, and UDP traffic is processed in SNAT mode.

    Default: `Hybrid`

    Allowed values: `SNAT`, `Hybrid`, `DSR`
  - **settings.createNodeRoutes**
    boolean

    Create routes to Pods on other Nodes.

    All Nodes must be located in the same L2 domain.
  - **settings.debugLogging**
    boolean

    Enables debug logging for Cilium components.

    Default: `false`
  - **settings.encryption**
    object

    CNI Cilium encryption settings.

    - **settings.encryption.mode**
      string

      Encryption mode:

      - `Disabled` — encryption is disabled.
      - `WireGuard` — enables traffic encryption between pods using WireGuard.

        > **i** Note! Before using WireGuard encryption, please ensure that the Linux kernel running on the cluster nodes has support WireGuard (either via the CONFIG_WIREGUARD=m kernel config option on Linux 5.6 and newer, or using an external WireGuard module for older kernels).

        > **i** Note! Using WireGuard may cause significant CPU consumption by Cilium agent pods.

      Default: `Disabled`

Allowed values: `Disabled`, `WireGuard`

- **settings.encryption.nodeToNodeModeEnabled**
  boolean

  Enabling traffic encryption between nodes and pods in HostNetwork.

  > **i** Note! The node-to-node traffic from and to the control-plane nodes is left unencrypted to prevent possible communication issues when renewing WireGuard public keys. At the same time, the traffic between control-plane components is still encrypted at the application layer.

  > **i** Note! When this mode is active, `NodePort` and `HostPort` are not available from the `PodNetwork`.

  Default: `false`

- **settings.exclusiveCNIPlugin**
  boolean

  Prevents additional CNI plugins from being enabled on cluster nodes. If the Istio module's `dataPlane.trafficRedirectionSetupMode` setting is set to `CNIPlugin`, the option will be automatically set to `false`.

  Explicitly disabling this option is recommended only when using an additional CNI or a specific CNI plugin in the cluster. An example of such a use case could be the Linkerd CNI plugin.

  Default: `true`

- **settings.extraLoadBalancerAlgorithmsEnabled**
  boolean

  Allows selectively overriding the load balancing algorithm using eBPF between backends for individual services. The annotation `service.cilium.io/lb-algorithm` is used for overriding with one of the following values: `random`, `maglev`, or `least-conn`.

  > **i** Note! This option requires the Linux kernel version on the nodes to be at least 5.15.

  Default: `false`

- **settings.labelsRegex**
  array of strings

  Cilium creates security identifiers based on the k8s entity labels. The more labels are involved in this process, the higher the access granularity that can be achieved. However, in large clusters, excessive granularity can create a heavy load. You can use the `labelsRegex` option to explicitly specify which labels are to be used for security policies, and which labels are to be neglected. Refer to the documentation to learn more about reducing the identity cardinality

  Each label must be specified by a RegExp expression in YAML quoted string format. Note that special characters must be escaped.

  Example:

  ```
  labelsRegex:
  - k8s:!app\.kubernetes\.io
  - k8s:io\.cilium\.k8s\.policy
  ```

- **settings.loadBalancerICMPReplyEnabled**
  boolean

  Enable ICMP Echo Reply generation for LoadBalancer service IPs.

  When enabled, nodes respond to ICMP Echo Requests (ping) addressed to LoadBalancer external IPs.

  Default: `false`

- **settings.loadBalancerSourceRangeAllTypes**
  boolean

  Enable loadBalancerSourceRanges CIDR filtering for all service types, not just LoadBalancer services. The corresponding NodePort and ClusterIP will also apply the CIDR filter. For more detailed information, please visit the link.

  Default: `true`

- **settings.masqueradeMode**
  string

  Cilium masquerade work mode for pods traffic leaving the cluster.

  - `BPF` - use cilium BPF. Basic operation mode.

    In this mode, masquerade will not be used if the destination IP address is within the `podSubnetCIDR` or `InternalIP`/`ExternalIP` of any of the cluster nodes.

  - `Netfilter` - use kernel Netfilter(iptables/nf_tables).

    In this mode, masquerade will not be used if the destination IP address is within the `podSubnetCIDR` or `InternalIP`/`ExternalIP` of any of the cluster nodes.

  > **i** In `BPF` mode, if `ExternalIP` other than `InternalIP` is specified for a cluster node, and that IP address is not currently assigned to any interface on the node, it will not be reachable from neighboring nodes from pods network. The `Netfilter` mode allows you to bypass this limitation, but some other features stop working, such as `EgressGateway`.

  Default: `BPF`

  Allowed values: `Netfilter`, `BPF`

- **settings.policyAuditMode**
  boolean

  Do not enforce any Network Policy. Just log allowed/disallowed connections.

  Default: `false`

- **settings.resourcesManagement**

  Settings for CPU and memory requests and limits by cilium agent pods.

  Examples:

  ```
  resourcesManagement:
    mode: VPA
    vpa:
      mode: InPlaceOrRecreate
      cpu:
        min: 50m
        max: 2
        limitRatio: 1.5
      memory:
        min: 256Mi
        max: 2Gi
        limitRatio: 1.5
  ```

  ```
  resourcesManagement:
    mode: Static
    static:
      requests:
        cpu: 55m
        memory: 256Mi
      limits:
        cpu: 2
        memory: 2Gi
  ```

```
resourcesManagement:
 mode: VPA
 vpa:
   mode: Auto
   cpu:
     min: 50m
     max: 2
     limitRatio: 1.5
   memory:
     min: 256Mi
     max: 2Gi
     limitRatio: 1.5
```

- **settings.resourcesManagement.mode**
  string

  Resource management mode:

  - `Static` is a classic one. In it, you specify requests/limits. The parameters of this mode are defined in the static parameter section;
  - `VPA` mode uses VPA. You can configure this mode by modifying parameters in the vpa parameter section.

  Default: `VPA`

  Allowed values: `VPA`, `Static`

- **settings.resourcesManagement.static**
  object

  Resource management options for the `Static` mode.

  - **settings.resourcesManagement.static.limits**
    object

    Configuring CPU and memory limits.

    - **settings.resourcesManagement.static.limits.cpu**
      string or number

      Configuring CPU limits.

      Pattern: `^[0-9]+m?$`

    - **settings.resourcesManagement.static.limits.memory**
      string or number

      Configuring memory limits.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.resourcesManagement.static.requests**
    object

    Resource requests settings for pods.

    - **settings.resourcesManagement.static.requests.cpu**
      string or number

      Configuring CPU requests.

      Pattern: `^[0-9]+m?$`

    - **settings.resourcesManagement.static.requests.memory**
      string or number

      Configuring memory requests.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.vpa**
  object

  Resource management options for the `VPA` mode.

  - **settings.resourcesManagement.vpa.cpu**
    object

    CPU-related VPA settings.

    - **settings.resourcesManagement.vpa.cpu.limitRatio**
      number

The CPU limits/requests ratio.

This ratio is used for calculating the initial CPU limits for a pod.

If this parameter is set, the VPA will recalculate the CPU limits while maintaining the specified limits/requests ratio.

- **settings.resourcesManagement.vpa.cpu.max**
  string or number

  The maximum value that the VPA can set for the CPU requests.

  Default: `4`

  Pattern: `^[0-9]+m?$`

- **settings.resourcesManagement.vpa.cpu.min**
  string or number

  The minimum value that the VPA can set for the CPU requests.

  Default: `100m`

  Pattern: `^[0-9]+m?$`

- **settings.resourcesManagement.vpa.memory**
  object

  Memory-related VPA settings.

  - **settings.resourcesManagement.vpa.memory.limitRatio**
    number

    The memory limits/requests ratio.

    This ratio is used for calculating the initial memory limits for a pod.

    If this parameter is set, the VPA will recalculate the memory limits while maintaining the specified limits/requests ratio.

  - **settings.resourcesManagement.vpa.memory.max**
    string or number

    The maximum memory requests the VPA can set.

    Default: `4Gi`

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.resourcesManagement.vpa.memory.min**
    string or number

    The minimum memory requests the VPA can set.

    Default: `128Mi`

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.vpa.mode**
  string

  VPA operating mode.

  - `Initial` — VPA sets initial values for pod resource requests and limits at pod creation time. Resource values are not changed automatically afterwards.
  - `InPlaceOrRecreate` — VPA attempts to update pod resources in place when supported by the cluster. If in-place updates are not possible, the pod will be recreated.
  - `Auto` — VPA automatically recreates pods to apply updated resource values.

    > **i** Starting from Deckhouse version 1.75, this mode is considered legacy. It is recommended to use `InPlaceOrRecreate` instead.

  Default: `Initial`

  Allowed values: `Initial`, `InPlaceOrRecreate`, `Auto`

- **settings.sctpEnabled**
  boolean

  Enable SCTP support for Cilium (for both incoming and outgoing traffic).

> **Note.** Cilium implements only basic SCTP support. Therefore, when using SCTP with Cilium, please be aware of the following limitations:
>
> - The following features are not currently supported: `SCTP multi-homing`, `NetworkPolicies for pod-to-VIP`.
> - For incoming SCTP packets port overwriting is not supported. This means that applications must be published through either HostPort or Service, where the `targetPort` MUST match the `port` specified. Otherwise, the packet will be dropped.
> - For outgoing SCTP packets, only 1:1 SNAT is supported by default. In this mode, only the src_ip is replaced, while the src_port remains unchanged. If the src_port is already in use on the node, the packet will be rejected. To enable src_port substitution, you must switch the `masqueradeMode` parameter to `Netfilter` mode.

Default: `false`

- **settings.svcSourceRangeCheck**
  boolean

  For services of the `loadBalancer` type, enables checking the source IP for compliance with loadBalancer.sourceRanges. The check is not performed if access is made from within the cluster or the `sourceRanges` parameter is empty/not specified. Some cloud providers support filtering of incoming traffic based on the `sourceRanges` parameter in their loadBalancer implementations. In such cases, the cilium documentation recommends disabling the `svcSourceRangeCheck` option.

  Do not enable if another balancer or any type of SNAT is running in front of the services.

  Default: `false`

- **settings.tunnelMode**
  string

  Tunnel mode.

  > Note! After changing the parameter, it is necessary to restart all nodes, otherwise, there may be problems with the availability of Pods!

  > Note! VXLAN mode is not compatible with following bpfLBMode modes: Hybrid, DSR. bpfLBMode will be overriden to SNAT if the tunnelMode is VXLAN.

  Default: `Disabled`

  Allowed values: `Disabled`, `VXLAN`

Custom Resources

> The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

EgressGatewayPolicy

Scope: Cluster
Version: v1alpha1

Interface for configuring policies to redirect application traffic to specific egress gateways described through the EgressGateway interface.

- **spec**
  object

  Required value

  - **spec.destinationCIDRs**
    array of strings

    Available in editions: SE+, EE

    Network requests towards these subnets will be routed through the egress gateway.

  - **spec.egressGatewayName**

string

Required value

Available in editions: SE+, EE

The name of the EgressGateway resource that describes the egress gateway for handling traffic.

- **spec.excludedCIDRs**
  array of strings

  Available in editions: SE+, EE

  Network ranges that are required to be excluded from the ranges described in `destinationCIDRs`. Note that you do not need to exclude the special cluster ranges (`podSubnetCIDR` and `serviceSubnetCIDR`).

- **spec.selectors**
  array of objects

  Available in editions: SE+, EE

  Application Pod Selectors. Their traffic will be routed through the appropriate egress gateway. For namespase designation use the `io.kubernetes.pod.namespace` pseudo-label.

  - **spec.selectors.podSelector**
    object

    Required value

    Available in editions: SE+, EE

    A label selector is a label query over a set of resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.

    - **spec.selectors.podSelector.matchExpressions**
      array of objects

      Available in editions: SE+, EE

      matchExpressions is a list of label selector requirements. The requirements are ANDed.

      A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

      - **spec.selectors.podSelector.matchExpressions.key**
        string

        Required value

        Available in editions: SE+, EE

        key is the label key that the selector applies to.

      - **spec.selectors.podSelector.matchExpressions.operator**
        string

        Required value

        Available in editions: SE+, EE

        operator represents a key’s relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

      - **spec.selectors.podSelector.matchExpressions.values**
        array of strings

        Available in editions: SE+, EE

        values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

    - **spec.selectors.podSelector.matchLabels**
      object

      Available in editions: SE+, EE

      matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are ANDed.

- **status**
  object

EgressGateway

Scope: Cluster
Version: v1alpha1

Interface for configuring a fault-tolerant egress gateway based on a group of nodes.

The configured egress gateways can be used in EgressGatewayPolicy interface to configure egress gateway for application Pods.

- **spec**
  object
  - **spec.nodeSelector**
    object

    Available in editions: SE+, EE

    The selector for a group of nodes that will transfer network requests to external services.

    Among these nodes, eligible nodes will be detected and one of them will be assigned as the active one. Signs of an eligible node:

    - The node is in Ready state.
    - Node is not cordoned.
    - The cilium-agent on the node is in the Ready state.

    Different EgressGateways can use common nodes for operation, and active nodes will be selected independently for each EgressGateway, thus distributing the load between them.

  - **spec.sourceIP**
    object
    - **spec.sourceIP.mode**
      string

      Required value

      Available in editions: SE+, EE

      A method for determining the source IP address to be assigned to requests through this gateway.

      Possible options:

      - `PrimaryIPFromEgressGatewayNodeInterface` (basic mode) — the primary IP address on the *public* network interface of the node will be used as the source IP address.

        Particularities:

        - When the active node goes down and a new node is assigned, the outgoing IP address will change.
        - The network subsystem on all egress nodes must be configured in advance:
          - the *public* interfaces must have the same name (eth1, for example),
          - all necessary routes for access to all external public services must be configured.

      - `VirtualIPAddress` (Virtual IP mode) — explicitly specify the outgoing IP address.

        Particularities:

        - If the active node goes down, the outgoing IP address will not change.
        - The network subsystem on all egress nodes must be configured in advance:
          - all necessary routes for access to all external public services must be configured,
          - the *public* interface must be prepared to automatically configure a *virtual* IP as a secondary IP address (if an egress node is designated as active, the address will not appear in the IP list on the *public* interface, but the node will emulate it with ARP-responses)

      Allowed values: `VirtualIPAddress`, `PrimaryIPFromEgressGatewayNodeInterface`

    - **spec.sourceIP.primaryIPFromEgressGatewayNodeInterface**
      object

      Used for basic mode (if `mode: PrimaryIPFromEgressGatewayNodeInterface`).

      - **spec.sourceIP.primaryIPFromEgressGatewayNodeInterface.interfaceName**
        string

        Available in editions: SE+, EE

        The interface name on the egress nodes through which external requests are processed.

    - **spec.sourceIP.virtualIPAddress**
      object

      Used for mode with Virtual IP (if `mode: PrimaryIPFromEgressGatewayNodeInterface`).

      - **spec.sourceIP.virtualIPAddress.interfaces**

array of strings

Available in editions: SE+, EE

The list of network interfaces to which the *virtual* IP will be simulated.

Default: `[]`

- **spec.sourceIP.virtualIPAddress.ip**
  string

  Available in editions: SE+, EE

  The *virtual* source IP address to be assigned to outgoing requests through the egress gateway.

- **status**
  object
  - **status.activeNodeName**
    string

    The name of the current active node.

  - **status.conditions**
    array of objects
    - **status.conditions.lastHeartbeatTime**
      string
    - **status.conditions.lastTransitionTime**
      string

      Required value

      lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

    - **status.conditions.message**
      string

      Required value

      message is a human readable message indicating details about the transition. This may be an empty string.

      Maximum length: `32768`

    - **status.conditions.observedGeneration**
      integer

      observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

      Allowed values: `0 <= X`

    - **status.conditions.reason**
      string

      Required value

      reason contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

      Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

      Length: `1..1024`

    - **status.conditions.status**
      string

      Required value

      status of the condition, one of True, False, Unknown.

      Allowed values: `True`, `False`, `Unknown`

    - **status.conditions.type**
      string

      Required value

      Type of condition in CamelCase or in foo.example.com/CamelCase.

Many .condition.type values are consistent across resources like Available, but because arbitrary conditions can be useful (see .node.status.conditions), the ability to deconflict is important. The regex it matches is (dns1123SubdomainFmt/)?(qualifiedNameFmt)

Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

Maximum length: `316`

- **status.observedGeneration**
  integer
- **status.readyNodes**
  integer

  Number of nodes ready to perform egress gateway functions.

HubbleMonitoringConfig

Scope: Cluster
Version: v1alpha1

- **spec**
  object

  Required value

  - **spec.extendedMetrics**
    object

    Configuration for exporting extended metrics from Cilium agents.

    Extended metrics provide deeper insight into cluster network behavior.

    > **i** **Warning.** Changing this parameter will trigger a restart of the Cilium agents.

    For more details, see the Cilium documentation.

    Example:

    ```
    extendedMetrics:
      enabled: true
      collectors:
      - name: drop
        contextOptions:
    labelsContext=source_ip,source_namespace,source_pod,destination_ip,destination_name
    space,destination_pod
      - name: flow
    ```

    - **spec.extendedMetrics.collectors**
      array of objects

      List of Hubble extended metrics to export.

      Each collector represents a specific metric type.

      For more details, see the Cilium documentation.

      Example:

      ```
      collectors:
      - name: drop
        contextOptions:
      labelsContext=source_ip,source_namespace,source_pod,destination_ip,destination
      _namespace,destination_pod
      - name: flow
      ```

      - **spec.extendedMetrics.collectors.contextOptions**
        string

        Optional context options for Hubble metrics.

        Defines which parameters are used to break down the metrics produced by a collector. If you do not specify `contextOptions`, the collector returns the default set of metrics aggregated overall, without breakdown by individual parameters. For example, for the `drop` collector, an empty `contextOptions` returns overall drop statistics without distinguishing the exact source and destination. When you set `contextOptions`, metric

values are grouped by the selected dimensions, for example by source and destination Pods, namespaces, and labels.

> **Warning.** A large number of labels on metrics can lead to Prometheus performance degradation.

For more information about supported options, see the Cilium documentation.

- **spec.extendedMetrics.collectors.name**
  string

  Name of the extended metric to export.

  Allowed values: `dns`, `drop`, `flow`, `flows-to-world`, `http`, `httpV2`, `icmp`, `kafka`, `tcp`

- **spec.extendedMetrics.enabled**
  boolean

  Enables the export of extended metrics from Cilium agents.

  Default: `false`

- **spec.flowLogs**
  object

  Configuration for exporting event logs from Cilium agents.

  Event logs provide detailed visibility into Cilium's network events.

  > **Warning.** Changing this parameter will trigger a restart of the Cilium agents.

  For more details see the Cilium documentation.

  Example:

  ```
  flowLogs:
    enabled: true
    allowFilterList:
    - verdict:
      - DROPPED
      - ERROR
    denyFilterList:
    - source_pod:
      - kube-system/
    - destination_pod:
      - kube-system/
    fieldMaskList:
    - time
    - source.namespace
    - verdict
    fileMaxSizeMB: 50
  ```

  - **spec.flowLogs.allowFilterList**
    array of objects

    Defines filters that control which events are exported for logging. Only events that match at least one of the specified filter objects will be exported. If no filters are specified, all events are allowed. Logical semantics: - Within a single filter object, all specified fields are combined using AND. - Across filter objects in the array, filters are combined using OR. For more details, see the Cilium documentation.

    - **spec.flowLogs.allowFilterList.destination_cluster_name**
      array of strings

      Destination cluster names.

    - **spec.flowLogs.allowFilterList.destination_fqdn**
      array of strings

      Destination fully qualified domain names (FQDN).

      - **Element of the array**
        string

        Maximum length: `255`

- **spec.flowLogs.allowFilterList.destination_identity**
  array of integers

  Destination security identity.

  - **Element of the array**
    integer

    Allowed values: `0 <= X <= 4294967295`

- **spec.flowLogs.allowFilterList.destination_ip**
  array of strings

  Destination IP or CIDR.

- **spec.flowLogs.allowFilterList.destination_label**
  array of strings

  Destination label selectors.

  For more information, please see the Cilium documentation.

  Examples:

  ```
  destination_label: k8s:role=frontend
  ```

  ```
  destination_label: reserved:kube-apiserver
  ```

- **spec.flowLogs.allowFilterList.destination_pod**
  array of strings

  Destination Pod name prefixes in the form NamespacedName (`namespace/pod`).

  Examples:

  ```
  destination_pod: kube-system/d8-kube-dns-7bc4648cb4-94tps
  ```

  ```
  destination_pod: kube-system/
  ```

  ```
  destination_pod: "/d8-kube-dns-7bc4648cb4-94tps"
  ```

- **spec.flowLogs.allowFilterList.destination_port**
  array of strings

  L4 destination port.

  - **Element of the array**
    string

    Pattern: `^(?:6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|[1-5]\d{4}|[1-9]\d{0,3})$`

- **spec.flowLogs.allowFilterList.destination_service**
  array of strings

  Destination service name prefixes in the form NamespacedName (`namespace/service`).

  Examples:

  ```
  destination_service: kube-system/kube-dns
  ```

  ```
  destination_service: kube-system/
  ```

  ```
  destination_service: "/kube-dns"
  ```

- **spec.flowLogs.allowFilterList.destination_workload**
  array of objects

  Destination workloads.

  - **spec.flowLogs.allowFilterList.destination_workload.kind**
    string

    Workload kind.

  - **spec.flowLogs.allowFilterList.destination_workload.name**
    string

Workload name.

- **spec.flowLogs.allowFilterList.dns_query**
  array of strings

  DNS query regex patterns.

- **spec.flowLogs.allowFilterList.drop_reason_desc**
  array of strings

  Drop reason (only for DROPPED verdict).

  - **Element of the array**
    string

    Allowed values: `DROP REASON UNKNOWN`, `INVALID SOURCE MAC`, `INVALID DESTINATION MAC`, `INVALID SOURCE IP`, `POLICY DENIED`, `INVALID PACKET DROPPED`, `CT TRUNCATED OR INVALID HEADER`, `CT MISSING TCP ACK FLAG`, `CT UNKNOWN L4 PROTOCOL`, `CT CANNOT CREATE ENTRY FROM PACKET`, `UNSUPPORTED L3 PROTOCOL`, `MISSED TAIL CALL`, `ERROR WRITING TO PACKET`, `UNKNOWN L4 PROTOCOL`, `UNKNOWN ICMPV4 CODE`, `UNKNOWN ICMPV4 TYPE`, `UNKNOWN ICMPV6 CODE`, `UNKNOWN ICMPV6 TYPE`, `ERROR RETRIEVING TUNNEL KEY`, `ERROR RETRIEVING TUNNEL OPTIONS`, `INVALID GENEVE OPTION`, `UNKNOWN L3 TARGET ADDRESS`, `STALE OR UNROUTABLE IP`, `NO MATCHING LOCAL CONTAINER FOUND`, `ERROR WHILE CORRECTING L3 CHECKSUM`, `ERROR WHILE CORRECTING L4 CHECKSUM`, `CT MAP INSERTION FAILED`, `INVALID IPV6 EXTENSION HEADER`, `IP FRAGMENTATION NOT SUPPORTED`, `SERVICE BACKEND NOT FOUND`, `NO TUNNEL OR ENCAPSULATION ENDPOINT`, `FAILED TO INSERT INTO PROXYMAP`, `REACHED EDT RATE LIMITING DROP HORIZON`, `UNKNOWN CONNECTION TRACKING STATE`, `LOCAL HOST IS UNREACHABLE`, `NO CONFIGURATION AVAILABLE TO PERFORM POLICY DECISION`, `UNSUPPORTED L2 PROTOCOL`, `NO MAPPING FOR NAT MASQUERADE`, `UNSUPPORTED PROTOCOL FOR NAT MASQUERADE`, `FIB LOOKUP FAILED`, `ENCAPSULATION TRAFFIC IS PROHIBITED`, `INVALID IDENTITY`, `UNKNOWN SENDER`, `NAT NOT NEEDED`, `IS A CLUSTERIP`, `FIRST LOGICAL DATAGRAM FRAGMENT NOT FOUND`, `FORBIDDEN ICMPV6 MESSAGE`, `DENIED BY LB SRC RANGE CHECK`, `SOCKET LOOKUP FAILED`, `SOCKET ASSIGN FAILED`, `PROXY REDIRECTION NOT SUPPORTED FOR PROTOCOL`, `POLICY DENY`, `VLAN FILTERED`, `INVALID VNI`, `INVALID TC BUFFER`, `NO SID`, `MISSING SRV6 STATE`, `NAT46`, `NAT64`, `AUTH REQUIRED`, `CT NO MAP FOUND`, `SNAT NO MAP FOUND`, `INVALID CLUSTER ID`, `UNSUPPORTED PROTOCOL FOR DSR ENCAP`, `NO EGRESS GATEWAY`, `UNENCRYPTED TRAFFIC`, `TTL EXCEEDED`, `NO NODE ID`, `DROP RATE LIMITED`, `IGMP HANDLED`, `IGMP SUBSCRIBED`, `MULTICAST HANDLED`, `DROP HOST NOT READY`, `DROP EP NOT READY`, `DROP NO EGRESS IP`

- **spec.flowLogs.allowFilterList.event_type**
  array of objects

  Event types to filter on.

  - **spec.flowLogs.allowFilterList.event_type.match_sub_type**
    boolean

    Whether to match on `sub_type`.

  - **spec.flowLogs.allowFilterList.event_type.sub_type**
    integer

    Secondary event type.

  - **spec.flowLogs.allowFilterList.event_type.type**
    integer

    Primary event type.

- **spec.flowLogs.allowFilterList.http_header**
  array of objects

  HTTP headers to match.

  - **spec.flowLogs.allowFilterList.http_header.key**
    string

    Header key.

  - **spec.flowLogs.allowFilterList.http_header.value**
    string

    Header value.

- **spec.flowLogs.allowFilterList.http_method**

array of strings

HTTP methods.

- **Element of the array**
  string

  Allowed values: `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `TRACE`, `PATCH`

- **spec.flowLogs.allowFilterList.http_path**
  array of strings

  Regex list to match HTTP path.

- **spec.flowLogs.allowFilterList.http_status_code**
  array of strings

  HTTP status code prefixes.

  - **Element of the array**
    string

    Pattern: `^(?:[1-5]\+|[1-5][0-9]\+|[1-5][0-9]{2})$`

- **spec.flowLogs.allowFilterList.http_url**
  array of strings

  Regex list to match HTTP URL.

- **spec.flowLogs.allowFilterList.interface**
  array of objects

  Network interface where flow was observed.

  - **spec.flowLogs.allowFilterList.interface.index**
    integer

    Interface index.

    Allowed values: `0 <= X`

  - **spec.flowLogs.allowFilterList.interface.name**
    string

    Interface name.

- **spec.flowLogs.allowFilterList.ip_version**
  array of strings

  IP version.

  - **Element of the array**
    string

    Allowed values: `IP_NOT_USED`, `IPv4`, `IPv6`

- **spec.flowLogs.allowFilterList.node_labels**
  array of strings

  Node label selectors.

  Examples:

  ```
  node_labels: node-role.kubernetes.io/control-plane=
  ```

  ```
  node_labels: topology.kubernetes.io/ru-central1-a
  ```

- **spec.flowLogs.allowFilterList.node_name**
  array of strings

  Node name patterns.

- **spec.flowLogs.allowFilterList.protocol**
  array of strings

  L4 or L7 protocol.

- **spec.flowLogs.allowFilterList.source_cluster_name**
  array of strings

  Source cluster names.

- **spec.flowLogs.allowFilterList.source_fqdn**
  array of strings

Source fully qualified domain names (FQDN).

- **Element of the array**
  string

  Maximum length: `255`

- **spec.flowLogs.allowFilterList.source_identity**
  array of integers

  Source security identity.

  - **Element of the array**
    integer

    Allowed values: `0 <= X <= 4294967295`

- **spec.flowLogs.allowFilterList.source_ip**
  array of strings

  Source IP or CIDR.

- **spec.flowLogs.allowFilterList.source_ip_xlated**
  array of strings

  Post-translation (SNAT) source IP or CIDR.

- **spec.flowLogs.allowFilterList.source_label**
  array of strings

  Source label selectors.

  For more information, please see the Cilium documentation.

  Examples:

  ```
  source_label: k8s:role=frontend
  ```

  ```
  source_label: reserved:kube-apiserver
  ```

- **spec.flowLogs.allowFilterList.source_pod**
  array of strings

  Source Pod name prefixes in the form NamespacedName (`namespace/pod`).

  Examples:

  ```
  source_pod: kube-system/d8-kube-dns-7bc4648cb4-94tps
  ```

  ```
  source_pod: kube-system/
  ```

  ```
  source_pod: "/d8-kube-dns-7bc4648cb4-94tps"
  ```

- **spec.flowLogs.allowFilterList.source_port**
  array of strings

  L4 source port.

  - **Element of the array**
    string

    Pattern: `^(?:6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|[1-5]\d{4}|[1-9]\d{0,3})$`

- **spec.flowLogs.allowFilterList.source_service**
  array of strings

  Source service name prefixes in the form NamespacedName (`namespace/service`).

  Examples:

  ```
  source_service: kube-system/kube-dns
  ```

  ```
  source_service: kube-system/
  ```

  ```
  source_service: "/kube-dns"
  ```

- **spec.flowLogs.allowFilterList.source_workload**
  array of objects

  Source workloads.

  - **spec.flowLogs.allowFilterList.source_workload.kind**
    string

    Workload kind.

  - **spec.flowLogs.allowFilterList.source_workload.name**
    string

    Workload name.

- **spec.flowLogs.allowFilterList.tcp_flags**
  array of objects

  TCP flags to match.

  - **spec.flowLogs.allowFilterList.tcp_flags.ACK**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.CWR**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.ECE**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.FIN**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.NS**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.PSH**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.RST**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.SYN**
    boolean
  - **spec.flowLogs.allowFilterList.tcp_flags.URG**
    boolean
- **spec.flowLogs.allowFilterList.trace_id**
  array of strings

  Trace ID.

- **spec.flowLogs.allowFilterList.traffic_direction**
  array of strings

  Traffic direction.

  - **Element of the array**
    string

    Allowed values: `TRAFFIC_DIRECTION_UNKNOWN`, `INGRESS`, `EGRESS`

- **spec.flowLogs.allowFilterList.uuid**
  array of strings

  Filters by flow UUIDs.

- **spec.flowLogs.allowFilterList.verdict**
  array of strings

  Match by verdict.

  - **Element of the array**
    string

    Allowed values: `VERDICT UNKNOWN`, `FORWARDED`, `DROPPED`, `ERROR`, `AUDIT`, `REDIRECTED`, `TRACED`, `TRANSLATED`

- **spec.flowLogs.denyFilterList**
  array of objects

  Defines filters that control which events are explicitly excluded from logging. Any event that matches at least one of the specified filter objects will be denied from being exported. If no filters are specified, no events are denied. This allows you to refine the set of events that are logged by excluding certain types. Logical semantics: - Within a single filter object, all specified fields are combined using AND. - Across filter objects in the array, filters are combined using OR. For more details, see the Cilium documentation.

  - **spec.flowLogs.denyFilterList.destination_cluster_name**
    array of strings

    Destination cluster names.

- **spec.flowLogs.denyFilterList.destination_fqdn**
  array of strings

  Destination fully qualified domain names (FQDN).

  - **Element of the array**
    string

    Maximum length: `255`

- **spec.flowLogs.denyFilterList.destination_identity**
  array of integers

  Destination security identity.

  - **Element of the array**
    integer

    Allowed values: `0 <= X <= 4294967295`

- **spec.flowLogs.denyFilterList.destination_ip**
  array of strings

  Destination IP or CIDR.

- **spec.flowLogs.denyFilterList.destination_label**
  array of strings

  Destination label selectors.

  For more information, please see the Cilium documentation.

  Examples:

  ```
  destination_label: k8s:role=frontend
  ```

  ```
  destination_label: reserved:kube-apiserver
  ```

- **spec.flowLogs.denyFilterList.destination_pod**
  array of strings

  Destination Pod name prefixes in the form NamespacedName (`namespace/pod`).

  Examples:

  ```
  destination_pod: kube-system/d8-kube-dns-7bc4648cb4-94tps
  ```

  ```
  destination_pod: kube-system/
  ```

  ```
  destination_pod: "/d8-kube-dns-7bc4648cb4-94tps"
  ```

- **spec.flowLogs.denyFilterList.destination_port**
  array of strings

  L4 destination port.

  - **Element of the array**
    string

    Pattern: `^(?:6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|[1-5]\d{4}|[1-9]\d{0,3})$`

- **spec.flowLogs.denyFilterList.destination_service**
  array of strings

  Destination service name prefixes in the form NamespacedName (`namespace/service`).

  Examples:

  ```
  destination_service: kube-system/kube-dns
  ```

  ```
  destination_service: kube-system/
  ```

  ```
  destination_service: "/kube-dns"
  ```

- **spec.flowLogs.denyFilterList.destination_workload**
  array of objects

Destination workloads.

- **spec.flowLogs.denyFilterList.destination_workload.kind**
  string

  Workload kind.

- **spec.flowLogs.denyFilterList.destination_workload.name**
  string

  Workload name.

- **spec.flowLogs.denyFilterList.dns_query**
  array of strings

  DNS query regex patterns.

- **spec.flowLogs.denyFilterList.drop_reason_desc**
  array of strings

  Drop reason (only for DROPPED verdict).

  - **Element of the array**
    string

    Allowed values: `DROP_REASON_UNKNOWN`, `INVALID_SOURCE_MAC`, `INVALID_DESTINATION_MAC`, `INVALID_SOURCE_IP`, `POLICY_DENIED`, `INVALID_PACKET_DROPPED`, `CT_TRUNCATED_OR_INVALID_HEADER`, `CT_MISSING_TCP_ACK_FLAG`, `CT_UNKNOWN_L4_PROTOCOL`, `CT_CANNOT_CREATE_ENTRY_FROM_PACKET`, `UNSUPPORTED_L3_PROTOCOL`, `MISSED_TAIL_CALL`, `ERROR_WRITING_TO_PACKET`, `UNKNOWN_L4_PROTOCOL`, `UNKNOWN_ICMPV4_CODE`, `UNKNOWN_ICMPV4_TYPE`, `UNKNOWN_ICMPV6_CODE`, `UNKNOWN_ICMPV6_TYPE`, `ERROR_RETRIEVING_TUNNEL_KEY`, `ERROR_RETRIEVING_TUNNEL_OPTIONS`, `INVALID_GENEVE_OPTION`, `UNKNOWN_L3_TARGET_ADDRESS`, `STALE_OR_UNROUTABLE_IP`, `NO_MATCHING_LOCAL_CONTAINER_FOUND`, `ERROR_WHILE_CORRECTING_L3_CHECKSUM`, `ERROR_WHILE_CORRECTING_L4_CHECKSUM`, `CT_MAP_INSERTION_FAILED`, `INVALID_IPV6_EXTENSION_HEADER`, `IP_FRAGMENTATION_NOT_SUPPORTED`, `SERVICE_BACKEND_NOT_FOUND`, `NO_TUNNEL_OR_ENCAPSULATION_ENDPOINT`, `FAILED_TO_INSERT_INTO_PROXYMAP`, `REACHED_EDT_RATE_LIMITING_DROP_HORIZON`, `UNKNOWN_CONNECTION_TRACKING_STATE`, `LOCAL_HOST_IS_UNREACHABLE`, `NO_CONFIGURATION_AVAILABLE_TO_PERFORM_POLICY_DECISION`, `UNSUPPORTED_L2_PROTOCOL`, `NO_MAPPING_FOR_NAT_MASQUERADE`, `UNSUPPORTED_PROTOCOL_FOR_NAT_MASQUERADE`, `FIB_LOOKUP_FAILED`, `ENCAPSULATION_TRAFFIC_IS_PROHIBITED`, `INVALID_IDENTITY`, `UNKNOWN_SENDER`, `NAT_NOT_NEEDED`, `IS_A_CLUSTERIP`, `FIRST_LOGICAL_DATAGRAM_FRAGMENT_NOT_FOUND`, `FORBIDDEN_ICMPV6_MESSAGE`, `DENIED_BY_LB_SRC_RANGE_CHECK`, `SOCKET_LOOKUP_FAILED`, `SOCKET_ASSIGN_FAILED`, `PROXY_REDIRECTION_NOT_SUPPORTED_FOR_PROTOCOL`, `POLICY_DENY`, `VLAN_FILTERED`, `INVALID_VNI`, `INVALID_TC_BUFFER`, `NO_SID`, `MISSING_SRV6_STATE`, `NAT46`, `NAT64`, `AUTH_REQUIRED`, `CT_NO_MAP_FOUND`, `SNAT_NO_MAP_FOUND`, `INVALID_CLUSTER_ID`, `UNSUPPORTED_PROTOCOL_FOR_DSR_ENCAP`, `NO_EGRESS_GATEWAY`, `UNENCRYPTED_TRAFFIC`, `TTL_EXCEEDED`, `NO_NODE_ID`, `DROP_RATE_LIMITED`, `IGMP_HANDLED`, `IGMP_SUBSCRIBED`, `MULTICAST_HANDLED`, `DROP_HOST_NOT_READY`, `DROP_EP_NOT_READY`, `DROP_NO_EGRESS_IP`

- **spec.flowLogs.denyFilterList.event_type**
  array of objects

  Event types to filter on.

  - **spec.flowLogs.denyFilterList.event_type.match_sub_type**
    boolean

    Whether to match on `sub_type`.

  - **spec.flowLogs.denyFilterList.event_type.sub_type**
    integer

    Secondary event type.

  - **spec.flowLogs.denyFilterList.event_type.type**
    integer

    Primary event type.

- **spec.flowLogs.denyFilterList.http_header**
  array of objects

  HTTP headers to match.

  - **spec.flowLogs.denyFilterList.http_header.key**

string

Header key.

- **spec.flowLogs.denyFilterList.http_header.value**
  string

  Header value.

- **spec.flowLogs.denyFilterList.http_method**
  array of strings

  HTTP methods.

  - **Element of the array**
    string

    Allowed values: `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `TRACE`, `PATCH`

- **spec.flowLogs.denyFilterList.http_path**
  array of strings

  Regex list to match HTTP path.

- **spec.flowLogs.denyFilterList.http_status_code**
  array of strings

  HTTP status code prefixes.

  - **Element of the array**
    string

    Pattern: `^(?:[1-5]\+|[1-5][0-9]\+|[1-5][0-9]{2})$`

- **spec.flowLogs.denyFilterList.http_url**
  array of strings

  Regex list to match HTTP URL.

- **spec.flowLogs.denyFilterList.interface**
  array of objects

  Network interface where flow was observed.

  - **spec.flowLogs.denyFilterList.interface.index**
    integer

    Interface index.

    Allowed values: `0 <= X`

  - **spec.flowLogs.denyFilterList.interface.name**
    string

    Interface name.

- **spec.flowLogs.denyFilterList.ip_version**
  array of strings

  IP version.

  - **Element of the array**
    string

    Allowed values: `IP_NOT_USED`, `IPv4`, `IPv6`

- **spec.flowLogs.denyFilterList.node_labels**
  array of strings

  Node label selectors.

  Examples:

  ```
  node_labels: node-role.kubernetes.io/control-plane=
  ```

  ```
  node_labels: topology.kubernetes.io/ru-central1-a
  ```

- **spec.flowLogs.denyFilterList.node_name**
  array of strings

  Node name patterns.

- **spec.flowLogs.denyFilterList.protocol**
  array of strings

L4 or L7 protocol.

- **spec.flowLogs.denyFilterList.source_cluster_name**
  array of strings

  Source cluster names.

- **spec.flowLogs.denyFilterList.source_fqdn**
  array of strings

  Source fully qualified domain names (FQDN).

  - **Element of the array**
    string

    Maximum length: `255`

- **spec.flowLogs.denyFilterList.source_identity**
  array of integers

  Source security identity.

  - **Element of the array**
    integer

    Allowed values: `0 <= X <= 4294967295`

- **spec.flowLogs.denyFilterList.source_ip**
  array of strings

  Source IP or CIDR.

- **spec.flowLogs.denyFilterList.source_ip_xlated**
  array of strings

  Post-translation (SNAT) source IP or CIDR.

- **spec.flowLogs.denyFilterList.source_label**
  array of strings

  Source label selectors.

  For more information, please see the Cilium documentation.

  Examples:

  ```
  source_label: k8s:role=frontend
  ```

  ```
  source_label: reserved:kube-apiserver
  ```

- **spec.flowLogs.denyFilterList.source_pod**
  array of strings

  Source Pod name prefixes in the form NamespacedName (`namespace/pod`).

  Examples:

  ```
  source_pod: kube-system/d8-kube-dns-7bc4648cb4-94tps
  ```

  ```
  source_pod: kube-system/
  ```

  ```
  source_pod: "/d8-kube-dns-7bc4648cb4-94tps"
  ```

- **spec.flowLogs.denyFilterList.source_port**
  array of strings

  L4 source port.

  - **Element of the array**
    string

    Pattern: `^(?:6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|[1-5]\d{4}|[1-9]\d{0,3})$`

- **spec.flowLogs.denyFilterList.source_service**
  array of strings

  Source service name prefixes in the form NamespacedName (`namespace/service`).

  Examples:

```
source_service: kube-system/kube-dns
```

```
source_service: kube-system/
```

```
source_service: "/kube-dns"
```

- **spec.flowLogs.denyFilterList.source_workload**
  array of objects

  Source workloads.

  - **spec.flowLogs.denyFilterList.source_workload.kind**
    string

    Workload kind.

  - **spec.flowLogs.denyFilterList.source_workload.name**
    string

    Workload name.

- **spec.flowLogs.denyFilterList.tcp_flags**
  array of objects

  TCP flags to match.

  - **spec.flowLogs.denyFilterList.tcp_flags.ACK**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.CWR**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.ECE**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.FIN**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.NS**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.PSH**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.RST**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.SYN**
    boolean
  - **spec.flowLogs.denyFilterList.tcp_flags.URG**
    boolean
- **spec.flowLogs.denyFilterList.trace_id**
  array of strings

  Trace ID.

- **spec.flowLogs.denyFilterList.traffic_direction**
  array of strings

  Traffic direction.

  - **Element of the array**
    string

    Allowed values: `TRAFFIC_DIRECTION_UNKNOWN`, `INGRESS`, `EGRESS`

- **spec.flowLogs.denyFilterList.uuid**
  array of strings

  Filters by flow UUIDs.

- **spec.flowLogs.denyFilterList.verdict**
  array of strings

  Match by verdict.

  - **Element of the array**
    string

    Allowed values: `VERDICT UNKNOWN`, `FORWARDED`, `DROPPED`, `ERROR`, `AUDIT`, `REDIRECTED`, `TRACED`, `TRANSLATED`

- **spec.flowLogs.enabled**
  boolean

  Enables event logging for Cilium agents. The log file is located on the node’s filesystem at `/var/log/cilium/hubble/flow.log`.

Default: `false`

- **spec.flowLogs.fieldMaskList**
  array of strings

  List of fields that should be displayed in events.

  For more details, see the Cilium documentation.

  - **Element of the array**
    string

    Allowed values: `time`, `uuid`, `verdict`, `drop_reason_desc`, `auth_type`, `ethernet`, `ethernet.source`, `ethernet.destination`, `IP`, `IP.source`, `IP.source_xlated`, `IP.destination`, `IP.ipVersion`, `IP.encrypted`, `l4`, `l4.TCP`, `l4.TCP.source_port`, `l4.TCP.destination_port`, `l4.TCP.flags`, `l4.TCP.flags.FIN`, `l4.TCP.flags.SYN`, `l4.TCP.flags.RST`, `l4.TCP.flags.PSH`, `l4.TCP.flags.ACK`, `l4.TCP.flags.URG`, `l4.TCP.flags.ECE`, `l4.TCP.flags.CWR`, `l4.TCP.flags.NS`, `l4.UDP`, `l4.UDP.source_port`, `l4.UDP.destination_port`, `l4.ICMPv4`, `l4.ICMPv4.type`, `l4.ICMPv4.code`, `l4.ICMPv6`, `l4.ICMPv6.type`, `l4.ICMPv6.code`, `l4.SCTP`, `l4.SCTP.source_port`, `l4.SCTP.destination_port`, `source`, `source.ID`, `source.identity`, `source.cluster_name`, `source.namespace`, `source.labels`, `source.pod_name`, `source.workloads`, `source.workloads.name`, `source.workloads.kind`, `destination`, `destination.ID`, `destination.identity`, `destination.cluster_name`, `destination.namespace`, `destination.labels`, `destination.pod_name`, `destination.workloads`, `destination.workloads.name`, `destination.workloads.kind`, `Type`, `node_name`, `node_labels`, `source_names`, `destination_names`, `l7`, `l7.type`, `l7.latency_ns`, `l7.record`, `l7.record.dns`, `l7.record.dns.query`, `l7.record.dns.ips`, `l7.record.dns.ttl`, `l7.record.dns.cnames`, `l7.record.dns.observation_source`, `l7.record.dns.rcode`, `l7.record.dns.qtypes`, `l7.record.dns.rrtypes`, `l7.record.http`, `l7.record.http.code`, `l7.record.http.method`, `l7.record.http.url`, `l7.record.http.protocol`, `l7.record.http.headers`, `l7.record.http.headers.key`, `l7.record.http.headers.value`, `l7.record.kafka`, `l7.record.kafka.error_code`, `l7.record.kafka.api_version`, `l7.record.kafka.api_key`, `l7.record.kafka.correlation_id`, `l7.record.kafka.topic`, `is_reply`, `event_type`, `event_type.type`, `event_type.sub_type`, `traffic_direction`, `trace_observation_point`, `trace_reason`, `file`, `file.name`, `file.line`, `debug_capture_point`, `interface`, `interface.index`, `interface.name`, `proxy_port`, `trace_context`, `trace_context.parent`, `trace_context.parent.trace_id`, `sock_xlate_point`, `socket_cookie`, `cgroup_id`, `extensions`, `egress_allowed_by`, `egress_allowed_by.name`, `egress_allowed_by.namespace`, `egress_allowed_by.labels`, `egress_allowed_by.revision`, `egress_allowed_by.kind`, `ingress_allowed_by`, `ingress_allowed_by.name`, `ingress_allowed_by.namespace`, `ingress_allowed_by.labels`, `ingress_allowed_by.revision`, `ingress_allowed_by.kind`, `egress_denied_by`, `egress_denied_by.name`, `egress_denied_by.namespace`, `egress_denied_by.labels`, `egress_denied_by.revision`, `egress_denied_by.kind`, `ingress_denied_by`, `ingress_denied_by.name`, `ingress_denied_by.namespace`, `ingress_denied_by.labels`, `ingress_denied_by.revision`, `ingress_denied_by.kind`, `source_service`, `source_service.name`, `source_service.namespace`, `destination_service`, `destination_service.name`, `destination_service.namespace`

- **spec.flowLogs.fileMaxSizeMB**
  integer

  Size in MB at which to rotate the Hubble export file.

  Default: `10`

  Allowed values: `1 <= X <= 150`

Examples

> **i** The module lifecycle stage: General Availability

Egress Gateway

Operation principle

Configuring an egress gateway requires two custom resources:

1. EgressGateway— describes the group of nodes that perform the egress gateway function in hot-standby mode:
   - Among the group of nodes match the `spec.nodeSelector`, the eligible nodes will be selected. One of them will be assigned as the active gateway. The active node is selected in alphabetical order.

Attributes of an eligible node:

- The node is in `Ready` state.
  - The node is not in the maintenance state (i.e., it is not cordoned).
  - The `cilium-agent` on the node is in the `Ready` state.
- When using EgressGateway in `VirtualIP` mode, an agent is launched on the active node which emulates a “virtual” IP address using the ARP protocol. The status of this agent’s pod is also taken into account when determining the eligibility of a node.
- Different EgressGateways can use the same nodes for operation. The active node is selected independently for each EgressGateway, which allows for load balancing between them.

2. EgressGatewayPolicy — describes the policy for routing network requests from pods in the cluster to a specific egress gateway defined using EgressGateway.

Node maintenance

To perform maintenance on a node that is currently the active egress gateway, follow these steps:

1. Remove the node label to exclude it from the egress gateway candidate pool. Egress-label is the label specified in `spec.nodeSelector` of your EgressGateway.

   ```
   d8 k label node <node-name> <egress-label>-
   ```

2. Cordon the node to prevent new pods from starting:

   ```
   d8 k cordon <node-name>
   ```

   After this, Cilium will automatically select a new active node from the remaining candidates. Traffic will continue routing through the new gateway without interruption.

3. After maintenance is complete, return the node to service:

   ```
   d8 k uncordon <node-name>
   d8 k label node <node-name> <egress-label>=<value>
   ```

> **i** Note: Reapplying the label may cause the node to become active again (if it is first in alphabetical order among candidates). To avoid immediate failback, temporarily reduce the number of EgressGateway replicas or adjust priorities using additional labels.

Comparison with CiliumEgressGatewayPolicy

The CiliumEgressGatewayPolicy implies configuring only one node as an egress gateway. If it fails, there are no failover mechanisms and the network connection will be broken.

Egress Gateway configuration examples

EgressGateway in PrimaryIPFromEgressGatewayNodeInterface mode (basic mode)

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGateway
metadata:
  name: my-egressgw
spec:
  nodeSelector:
    dedicated/egress: ""
  sourceIP:
    mode: PrimaryIPFromEgressGatewayNodeInterface
    primaryIPFromEgressGatewayNodeInterface:
      # The "public" interface must have the same name on all nodes that matching the
nodeSelector.
      # If the active node fails, traffic will be redirected through the backup node and
      # the source IP address of the network packets will change.
      interfaceName: eth1
```

EgressGateway in VirtualIPAddress mode (Virtual IP mode)

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGateway
metadata:
  name: my-egressgw
spec:
  nodeSelector:
    dedicated/egress: ""
  sourceIP:
    mode: VirtualIPAddress
    virtualIPAddress:
      # Each node must have all the necessary routes configured to access all external public
services,
     # the "public" interface must be prepared for automatic configuration of the "virtual"
IP as a secondary IP address.
     # In case of failure of the active node, traffic will be redirected through the backup
node and
     # the source IP address of the network packets will not change.
     ip: 172.18.18.242
     # List of network interfaces for Virtual IP
     interfaces:
     - eth1
```

EgressGatewayPolicy

```
apiVersion: network.deckhouse.io/v1alpha1
kind: EgressGatewayPolicy
metadata:
  name: my-egressgw-policy
spec:
  destinationCIDRs:
  - 0.0.0.0/0
  egressGatewayName: my-egressgw
  selectors:
  - podSelector:
      matchLabels:
        app: backend
        io.kubernetes.pod.namespace: my-ns
```

HubbleMonitoringConfig

The cluster-scoped HubbleMonitoringConfig resource is intended to configure data export from Hubble, which runs inside Cilium agents.

HubbleMonitoringConfig configuration examples

Enabling extended metrics and flow logs export (with filters and field mask)

> ⚠ The HubbleMonitoringConfig resource **must be named** `hubble-monitoring-config`.

Example of enabling metrics and export:

```
apiVersion: deckhouse.io/v1alpha1
kind: HubbleMonitoringConfig
metadata:
  name: hubble-monitoring-config
spec:
  extendedMetrics:
    enabled: true
    collectors:
      - name: drop
        # Add additional labels context for the selected collector.
        contextOptions:
"labelsContext=source_ip,source_namespace,source_pod,destination_ip,destination_namespace,de
stination_pod"
      - name: flow
  flowLogs:
    enabled: true
    # Allow only the specified events to be written to the log file
/var/log/cilium/hubble/flow.log.
    allowFilterList:
      - verdict: ["DROPPED", "ERROR"]
    # Exclude events matching the denyFilterList from the log file.
    denyFilterList:
      - source_pod:
        - kube-system/
      - destination_pod:
        - kube-system/
    # Persist only the specified fields in each record.
    fieldMaskList:
      - time
      - verdict
    # Maximum log file size (in MB) before rotation.
    fileMaxSizeMB: 30
```

Collecting Hubble flow logs with the log-shipper module

To collect flow logs, use the `log-shipper` module.

Create a ClusterLoggingConfig resource that reads the log file from the node filesystem:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: cilium-hubble-flow-logs
spec:
  type: File
  file:
    include:
      - /var/log/cilium/hubble/flow.log
```

The cni-flannel module

> **i** The module lifecycle stage: General Availability

This module is responsible for providing a network between multiple nodes in a cluster using the flannel module.

Third-party components

List of third-party software used in the `cni-flannel` module:

- Flannel 0.27.4

  License: Apache License 2.0

  A simple and easy way to configure a layer 3 network fabric designed for Kubernetes.

Configuration

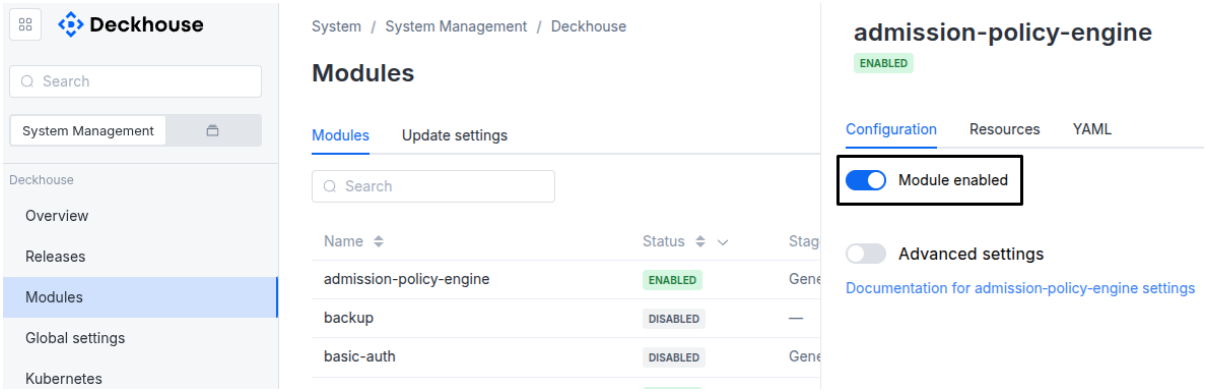
> **i** The module lifecycle stage: General Availability

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cni-flannel` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable cni-flannel
```

- **Using ModuleConfig `cni-flannel`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `cni-flannel` (create it if necessary);

  Example of a manifest to enable module `cni-flannel`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: cni-flannel
spec:
  enabled: true
```

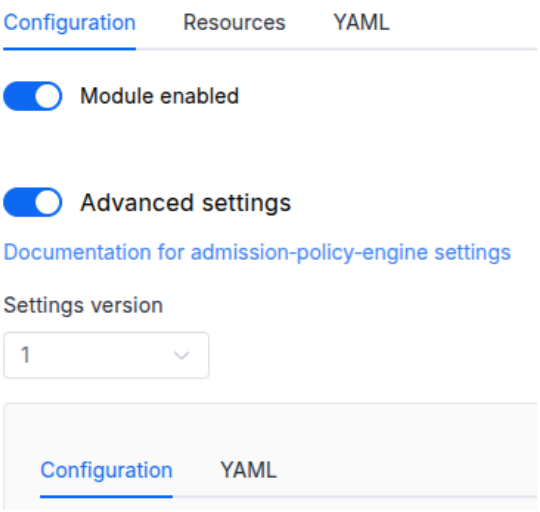
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `cni-flannel` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `cni-flannel` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `cni-flannel`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `cni-flannel` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc cni-flannel
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `cni-flannel` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `cni-flannel`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: cni-flannel
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.podNetworkMode**
    string

    Work mode.

    > **i** **NOTE!** All cluster nodes must be rebooted after switching work mode. You can change the parameter **only** in bare metal clusters.

    Default: `HostGW`

    Allowed values: `HostGW`, `VXLAN`

The cni-simple-bridge module

> **i** The module lifecycle stage: General Availability

The module does not have any settings.

It is automatically enabled for the following cloud providers:

- AWS
- Azure
- GCP

Managing control plane

> **i** The module lifecycle stage: General Availability

The `control-plane-manager` (CPM) module is responsible for managing the cluster's control plane components. It runs on all master nodes of the cluster (nodes that have the `node-role.kubernetes.io/control-plane: ""` label).

The control-plane-manager:

- **Manages certificates** required for the operation of the control plane (renews certificates and re-issues them in response to configuration changes, among other things). This feature allows the CPM to automatically maintain a secure control plane configuration and quickly add additional SANs for organizing secure access to the Kubernetes API.
- **Configures components**. The CPM module automatically creates the required configs and manifests of the control plane components.
- **Upgrades/downgrades components**. Makes sure that the versions of the components in the cluster are the same.
- **Manages the configuration of the etcd cluster** and its members. The CPM module scales master nodes and migrates the cluster from single-master to multi-master (and vice versa).
- **Configures kubeconfig**. The CPM module maintains up-to-date kubeconfig files on control-plane nodes. It generates, renews, and updates kubeconfigs for control-plane components and the admin kubeconfig (`admin.conf`). By default, it creates a symlink for the root user (`/root/.kube/config` -> `admin.conf`). When the user-authz module is enabled, the symlink can be turned off via the `rootKubeconfigSymlink` parameter in the **control-plane-manager** module (see FAQ). The CPM also hardens file permissions on `admin.conf` and `super-admin.conf`.
- **Extends scheduler functionality** by integrating external plugins via webhooks. Manages by KubeSchedulerWebhookConfiguration resource. Allows more complex logic to be used in workload scheduling tasks within the cluster. For example:
  - placing data storage application pods closer to the data itself,
  - state-based node prioritization (network load, storage subsystem status, etc.),
  - dividing nodes into zones, etc.

Managing certificates

The CPM module manages certificates of the `control-plane` components, such as:

- Server certificates for `kube-apiserver` & `etcd`. These are stored in the secret `d8-pki` of the namespace `kube-system`:
  - the root CA kubernetes certificate (`ca.crt` & `ca.key`),
  - the root CA etcd certificate (`etcd/ca.crt` & `etcd/ca.key`),
  - the RSA certificate and the key for signing Service Accounts (`sa.pub` & `sa.key`),
  - the root CA certificate for the extension API servers (`front-proxy-ca.key` & `front-proxy-ca.crt`).
- Client certificates for connecting `control-plane` components to each other. The CPM module issues, renews, and re-issues if something has changed (e.g., the SAN list). These certificates are stored on the nodes only:
  - The server-side API server certificate (`apiserver.crt` & `apiserver.key`).
  - The client-side certificate for connecting `kube-apiserver` to `kubelet` (`apiserver-kubelet-client.crt` & `apiserver-kubelet-client.key`).
  - The client-side certificate for connecting `kube-apiserver` to `etcd` (`apiserver-etcd-client.crt` & `apiserver-etcd-client.key`).
  - The client-side certificate for connecting `kube-apiserver` to the extension API servers (`front-proxy-client.crt` & `front-proxy-client.key`).
  - The server-side `etcd` certificate (`etcd/server.crt` & `etcd/server.key`).
  - The client-side certificate for connecting `etcd` to other cluster members (`etcd/peer.crt` & `etcd/peer.key`).
  - The client-side certificate for connecting `kubelet` to `etcd` for performing health-checks (`etcd/healthcheck-client.crt` & `etcd/healthcheck-client.key`).

Also, the CPM module lets you add the additional SANs to certificates (this way, you can quickly and effortlessly add more "entry points" to the Kubernetes API).

The CPM module automatically updates the kubeconfig configuration when certificates are changed.

Scaling

The CPM module supports `control plane` running in a *single-master* or *multi-master* mode.

In the *single-master* mode:

- `kube-apiserver` only uses the `etcd` instance that is hosted on the same node;
- A proxy server is configured on the node that responds to localhost, `kube-apiserver` responds to the IP address of the master node.

In the *multi-master* mode, `control plane` components are automatically deployed in a fault-tolerant manner:

- `kube-apiserver` is configured to work with all etcd instances;
- The additional proxy server that processes localhost requests is set up on each master node. By default, the proxy server sends requests to the local `kube-apiserver` instance. If it is unavailable, the proxy tries to connect to other `kube-apiserver` instances.

Scaling master nodes

The `control-plane` nodes are scaled automatically using the `node-role.kubernetes.io/control-plane=""` label:

- Attaching the `node-role.kubernetes.io/control-plane=""` label to a node results in deploying `control plane` components on this node, connecting the new `etcd` node to the etcd cluster, and regenerating all the necessary certificates and config files.
- Removing the `node-role.kubernetes.io/control-plane=""` label results in deleting all `control plane` components on a node, gracefully removing it from the etcd cluster, and regenerating all the necessary config files and certificates.

⚠ Manual `etcd` actions are required when decreasing the number of nodes from two to one. In all other cases, all the necessary actions are performed automatically. Please note that when scaling from any number of master nodes to one, sooner or later at the last step, the situation of scaling nodes from two to one will arise.

Dynamic terminated pod garbage collection threshold

Automatically configures the optimal `--terminated-pod-gc-threshold` based on cluster size:

- **Small clusters** (less than 100 nodes): 1000 terminated Pods.
- **Medium clusters** (from 100 to 300 nodes): 3000 terminated Pods.
- **Large clusters** (from 300 nodes): 6000 terminated Pods.

i Note. This feature only takes effect in environments where the `--terminated-pod-gc-threshold` parameter is configurable. On managed Kubernetes services (such as EKS, GKE, AKS), this setting is controlled by managed provider.

Version control

Patch versions of control plane components (i.e. within the minor version, for example, from `1.31.13` to `1.31.14`) are upgraded automatically together with the Deckhouse version updates. You can't manage patch version upgrades.

Upgrading **minor versions** of control plane components (e.g. from `1.31.*` to `1.32.*`) can be managed using the `kubernetesVersion` parameter. It specifies the automatic update mode (if set to `Automatic`) or the desired minor version of the control plane. The default control plane version (to use with `kubernetesVersion: Automatic`) as well as a list of supported Kubernetes versions can be found in the documentation.

The control plane upgrade is performed in a safe way for both single-master and multi-master clusters. The API server may be temporarily unavailable during the upgrade. At the same time, it does not affect the operation of applications in the cluster and can be performed without scheduling a maintenance window.

If the target version (set with the kubernetesVersion parameter) does not match the current control plane version in the cluster, a smart strategy for changing component versions is applied:

- General remarks
  - Updating in different NodeGroups is performed in parallel. Within each NodeGroup, nodes are updated sequentially, one at a time.
- When upgrading:
  - Upgrades are carried out sequentially, one minor version at a time: 1.31 -> 1.32, 1.32 -> 1.33, 1.33 -> 1.34.
  - At each step, the control plane version is upgraded first, followed by kubelet upgrades on the cluster nodes.
- When downgrading:
  - Successful downgrading is only guaranteed for a single version down from the maximum minor version of the control plane ever used in the cluster.
  - kubelets on the cluster nodes are downgraded first, followed by the control plane components.

Auditing

Kubernetes Auditing can help you if you need to keep track of operations in your Namespaces or troubleshoot the cluster. You can configure it by setting the appropriate Audit Policy. As the result you will have the log file `/var/log/kube-audit/audit.log` containing audit events according to the configured Policy.

By default, in a cluster with Deckhouse, a basic policy is created for logging events:

- related to the creation, deletion, and changing of resources;
- committed from the names of ServiceAccounts from the “system” Namespace `kube-system`, `d8-*`;
- committed with resources in the “system” Namespace `kube-system`, `d8-*`.

A basic policy can be disabled by setting the basicAuditPolicyEnabled flag to `false`.

When OIDC authentication is configured, additional user information is included in audit logs under the `user.extra` field:

- `user-authn.deckhouse.io/name` — user’s display name
- `user-authn.deckhouse.io/preferred username` — user’s preferred username
- `user-authn.deckhouse.io/dex-provider` — Dex provider identifier (requires `federated:id` scope)

You can find how to set up policies in a special FAQ section.

Feature Gates

You can configure feature gates using the enabledFeatureGates parameter of the `control-plane-manager` ModuleConfig.

Changing the list of feature gates causes a restart of the corresponding component (for example, `kube-apiserver`, `kube-scheduler`, `kube-controller-manager`, `kubelet`).

The following example enables the `ComponentFlagz` and `ComponentStatusz` feature gates:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    enabledFeatureGates:
      - ComponentFlagz
      - ComponentStatusz
```

If a feature gate is not supported or is deprecated, the monitoring system generates the D8ProblematicFeatureGateInUse alert indicating that the feature gate will not be applied.

> ⚠ The Kubernetes version update (controlled by the kubernetesVersion parameter) will not occur if the list of enabled feature gates for the new version of Kubernetes includes deprecated feature gates.

More information about feature gates is available in the Kubernetes documentation.

▼ List of available feature gates...

Kubernetes 1.31

- `AuthorizeNodeWithSelectors`
- `AuthorizeWithSelectors`
- `ConcurrentWatchObjectDecode`
- `CrossNamespaceVolumeDataSource`
- `DisableAllocatorDualWrite`
- `HPAScaleToZero`
- `ImageVolume`
- `InPlacePodVerticalScaling`
- `JobManagedBy`
- `KubeletPodResourcesGet`
- `MaxUnavailableStatefulSet`
- `MemoryQoS`
- `MutatingAdmissionPolicy`
- `OrderedNamespaceDeletion`
- `ProcMountType`
- `RecoverVolumeExpansionFailure`
- `RelaxedEnvironmentVariableValidation`
- `ResourceHealthStatus`

- `SELinuxMount`
- `SchedulerQueueingHints`
- `StrictCostEnforcementForVAP`
- `StrictCostEnforcementForWebhooks`

Kubernetes 1.32

- `AllowUnsafeMalformedObjectDeletion`
- `ComponentFlagz`
- `ComponentStatusz`
- `ConcurrentWatchObjectDecode`
- `CrossNamespaceVolumeDataSource`
- `DisableAllocatorDualWrite`
- `HPAScaleToZero`
- `ImageVolume`
- `InPlacePodVerticalScaling`
- `InPlacePodVerticalScalingAllocatedStatus`
- `KubeletPodResourcesGet`
- `MaxUnavailableStatefulSet`
- `MemoryQoS`
- `MutatingAdmissionPolicy`
- `OrderedNamespaceDeletion`
- `PodLifecycleSleepActionAllowZero`
- `PodLogsQuerySplitStreams`
- `ProcMountType`
- `RelaxedDNSSearchValidation`
- `ResourceHealthStatus`
- `SELinuxChangePolicy`
- `SELinuxMount`
- `SchedulerAsyncPreemption`

Kubernetes 1.33

- `AllowParsingUserUIDFromCertAuth`
- `AllowUnsafeMalformedObjectDeletion`
- `ComponentFlagz`
- `ComponentStatusz`
- `ConcurrentWatchObjectDecode`
- `ContainerStopSignals`
- `CrossNamespaceVolumeDataSource`
- `DeploymentReplicaSetTerminatingReplicas`
- `DisableAllocatorDualWrite`
- `HPAConfigurableTolerance`
- `HPAScaleToZero`
- `ImageVolume`
- `KubeletEnsureSecretPulledImages`
- `KubeletPSI`
- `KubeletPodResourcesGet`
- `ListFromCacheSnapshot`
- `MaxUnavailableStatefulSet`
- `MemoryQoS`
- `MutatingAdmissionPolicy`
- `PodLogsQuerySplitStreams`
- `PodObservedGenerationTracking`
- `PreferSameTrafficDistribution`
- `ResourceHealthStatus`
- `SELinuxMount`
- `StrictIPCIDRValidation`

Kubernetes 1.34

- `AllowUnsafeMalformedObjectDeletion`
- `ComponentFlagz`
- `ComponentStatusz`
- `ConcurrentWatchObjectDecode`
- `ContainerRestartRules`
- `ContainerStopSignals`
- `CrossNamespaceVolumeDataSource`
- `DeploymentReplicaSetTerminatingReplicas`
- `EnvFiles`
- `HPAConfigurableTolerance`
- `HPAScaleToZero`
- `HostnameOverride`
- `ImageVolume`
- `KubeletEnsureSecretPulledImages`
- `MaxUnavailableStatefulSet`
- `MemoryQoS`

- `MutatingAdmissionPolicy`
- `PodLogsQuerySplitStreams`
- `ResourceHealthStatus`
- `SELinuxMount`
- `StrictIPCIDRValidation`

Kubernetes 1.35

- `AllowUnsafeMalformedObjectDeletion`
- `CRDObservedGenerationTracking`
- `ComponentFlagz`
- `ComponentStatusz`
- `ConcurrentWatchObjectDecode`
- `ConstrainedImpersonation`
- `ContainerStopSignals`
- `CrossNamespaceVolumeDataSource`
- `HPAScaleToZero`
- `MemoryQoS`
- `MutablePVNodeAffinity`
- `MutablePodResourcesForSuspendedJobs`
- `MutableSchedulingDirectivesForSuspendedJobs`
- `MutatingAdmissionPolicy`
- `NodeDeclaredFeatures`
- `PodLogsQuerySplitStreams`
- `ResourceHealthStatus`
- `RestartAllContainersOnContainerExits`
- `SELinuxMount`
- `StrictIPCIDRValidation`
- `TaintTolerationComparisonOperators`

Protecting sensitive fields in custom resources

The `CRDSensitiveData` feature gate provides field-level protection for sensitive data in resources marked with the `x-kubernetes-sensitive-data: true` annotation. This feature is implemented as a patch to `kube-apiserver` (apiextensions-apiserver) and is supported starting from Kubernetes version 1.31.

The `x-kubernetes-sensitive-data` marker is validated by `kube-apiserver` when applying a resource:

- Marker requires the `CRDSensitiveData` feature gate to be enabled, which is enabled automatically when `apiserver.encryptionEnabled` is set to `true`.
- Marker can't be set on the root of the schema (the `openAPIV3Schema` node). To protect all fields of a resource, add the marker to the `spec` property (or a subtree below it), not to the schema root — the root also includes system fields (`apiVersion`, `kind`, `metadata`), which cannot be encrypted.
- Field type must be one of the OpenAPI v3 types: `string`, `integer`, `number`, `boolean`, `object`, or `array`. Applying the marker to `object` or `array` makes the entire subtree sensitive.
- Fields defined with `x-kubernetes-int-or-string: true` are supported.
- Marker is not allowed inside `anyOf`, `oneOf`, `allOf`, or `not` branches (this is enforced by the structural schema validator).

If at least one field in the resource schema is marked with `x-kubernetes-sensitive-data: true`, the following protection mechanisms are applied to all custom resources of this type:

- **Encryption in etcd**: Entire resource is encrypted using the same mechanism as Kubernetes Secrets. Requires enabling the `apiserver.encryptionEnabled` parameter.
- **RBAC-based field filtering**: For `get`, `list`, and `watch` requests, sensitive fields are removed from API responses if the caller does not have the corresponding permissions on the `<resource>/sensitive` subresource.
- **Audit log masking**: Values of sensitive fields are replaced with `"******"` in audit logs, regardless of RBAC permissions and audit level.

To enable sensitive data protection, set the `apiserver.encryptionEnabled` parameter to `true`. The `CRDSensitiveData` feature gate is enabled automatically and it shouldn't be specified manually:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    apiserver:
      encryptionEnabled: true
```

⚠ Enabling `encryptionEnabled` is irreversible and triggers a `kube-apiserver` restart.

For details, see the following sections:

- FAQ: Instructions for enabling sensitive data protection.
- Examples: Configuration examples and results.

Third-party components

List of third-party software used in the `control-plane-manager` module:

- Kubernetes

  License: Apache License 2.0

  An open source system for managing containerized applications across multiple hosts.

- Etcd 3.6.10

  License: Apache License 2.0

  A distributed reliable key-value store for the most critical data of a distributed system.

Configuration

i The module lifecycle stage: General Availability

Some cluster parameters that affect control plane management are derived from the ClusterConfiguration resource.

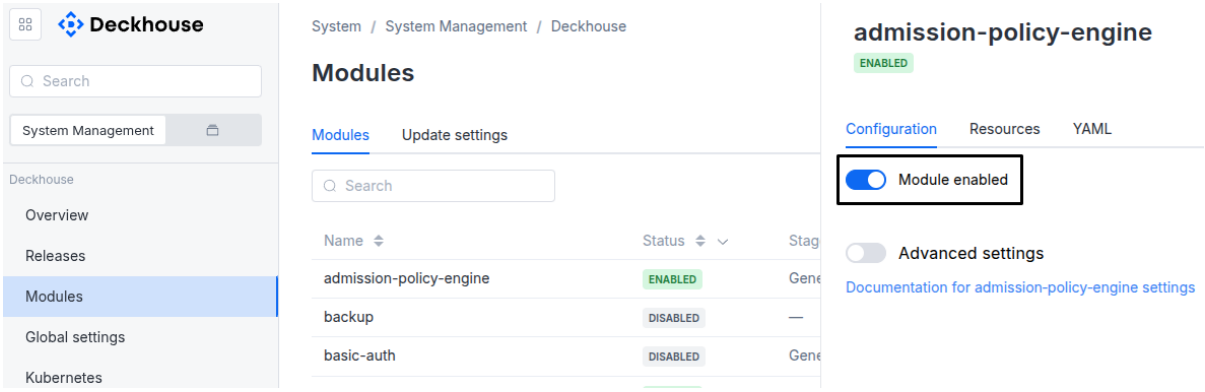
The module has 10 alerts.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `control-plane-manager` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable control-plane-manager
```

- **Using ModuleConfig `control-plane-manager`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `control-plane-manager` (create it if necessary);

  Example of a manifest to enable module `control-plane-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  enabled: true
```

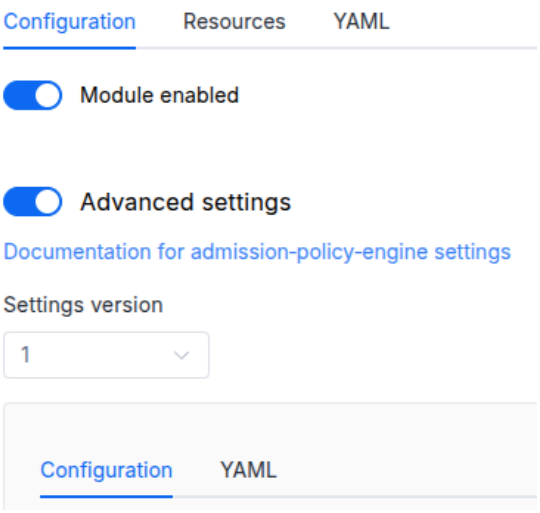
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `control-plane-manager` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `control-plane-manager` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `control-plane-manager`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `control-plane-manager` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc control-plane-manager
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `control-plane-manager` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `control-plane-manager`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: control-plane-manager
  spec:
    version: 2
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Delete `etcd.externalMembersNames`

Parameters

Schema version: 2

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    apiserver:
      bindToWildcard: true
      certSANs:
      - bakery.infra
      - devs.infra
      loadBalancer: {}
```

- **settings**
  object
  - **settings.apiserver**
    object

    `kube-apiserver` parameters.

    - **settings.apiserver.admissionPlugins**
      array of strings

      List of enabled additional admission plugins.

      Note, that in addition to the admission plugins enabled by default in Kubernetes, the following admission plugins are also always enabled:

      - `ExtendedResourceToleration`
      - `PodNodeSelector`
      - `PodTolerationRestriction`
      - `EventRateLimit` with the following config:

        ```
        apiVersion: eventratelimit.admission.k8s.io/v1alpha1
        kind: Configuration
        limits:
        - type: Namespace
          qps: 50
          burst: 100
          cacheSize: 2000
        ```

      > **i** Note that th `PodNodeSelector` admission plugin does not require specifiying a global configuration, it relies on annotated Namespaces.

      Example:

      ```
      admissionPlugins:
      - AlwaysPullImages
      - NamespaceAutoProvision
      ```

      - **Element of the array**
        string

Allowed values: `AlwaysPullImages`, `NamespaceAutoProvision`, `OwnerReferencesPermissionEnforcement`, `PodNodeSelector`, `PodTolerationRestriction`

- **settings.apiserver.auditLog**
  object

  Audit policy settings

  Default: `{"output":"File"}`

  - **settings.apiserver.auditLog.output**
    string

    Required value

    Audit logs target stream.

    Default: `File`

    Allowed values: `File`, `Stdout`

    Example:

    ```
    output: Stdout
    ```

  - **settings.apiserver.auditLog.path**
    string

    Directory path for logs if the output is “File”, otherwise ignored.

    Default: `/var/log/kube-audit`

    Pattern: `^[a-zA-Z0-9_/.-]+[a-zA-Z0-9_.-]$`

- **settings.apiserver.auditPolicyEnabled**
  boolean

  Set the audit policies using the configuration from the `kube-system/audit-policy` Secret.

  Default: `false`

- **settings.apiserver.authn**
  object

  Optional authentication parameters for Kubernetes API clients.

  By default, they are taken from user-authn module ConfigMap.

  - **settings.apiserver.authn.oidcCA**
    string

    OIDC provider CA.

  - **settings.apiserver.authn.oidcIssuerAddress**
    string

    OIDC provider network address alias.

    Examples:

    ```
    oidcIssuerAddress: 1.2.3.4
    ```

    ```
    oidcIssuerAddress: ''
    ```

  - **settings.apiserver.authn.oidcIssuerURL**
    string

    OIDC provider URL.

    Example:

    ```
    oidcIssuerURL: https://my-super-site.tech/
    ```

  - **settings.apiserver.authn.webhookCA**
    string

    Authorization webhook CA.

  - **settings.apiserver.authn.webhookCacheTTL**
    string

The duration to cache responses from the webhook token authenticator.

It is specified as a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

Example:

```
webhookCacheTTL: 5m
```

- **settings.apiserver.authn.webhookURL**
  string

  Authentication webhook URL.

  Example:

  ```
  webhookURL: https://127.0.0.1:40443/
  ```

- **settings.apiserver.authz**
  object

  Optional authorization parameters for Kubernetes API clients.

  By default, they are taken from user-authz module ConfigMap.

  - **settings.apiserver.authz.webhookCA**
    string

    Authorization webhook CA.

  - **settings.apiserver.authz.webhookURL**
    string

    Authorization webhook URL.

    Example:

    ```
    webhookURL: https://127.0.0.1:40443/
    ```

- **settings.apiserver.basicAuditPolicyEnabled**
  boolean

  Enforce basic Deckhouse audit policies.

  Default: `true`

- **settings.apiserver.bindToWildcard**
  boolean

  Specifies whether to listen on `0.0.0.0`.

  By default, the API server listens on the hostIP. The latter usually corresponds to the Internal node address; however, the actual IP depends on the cluster type (Static or Cloud) and the layout selected.

  Default: `false`

- **settings.apiserver.certSANs**
  array of strings

  Array of SANs, with which the API server certificate will be generated.

  In addition to the passed list, the following list is always used:

  - `kubernetes`;
  - `kubernetes.default`;
  - `kubernetes.default.svc`;
  - `kubernetes.default.svc.cluster.local`;
  - Kubernetes API server address;
  - `127.0.0.1`;
  - host name;
  - host IP.

  Example:

  ```
  certSANs:
  - my-site.com
  - 192.168.67.76
  ```

- **Element of the array**
  string

  Pattern: `^[0-9a-zA-Z\.-]+$`

- **settings.apiserver.encryptionEnabled**
  boolean

  Enables data encryption in etcd (see Encrypting secret data at rest).

  When enabled, a Secret `kube-system/d8-secret-encryption-key` is created to store the encryption key.

  Enabling this parameter automatically enables the `CRDSensitiveData` feature gate for `kube-apiserver`, which protects custom resource fields marked with `x-kubernetes-sensitive-data: true` via encryption in etcd, RBAC-based field filtering with the `<resource>/sensitive` subresource and hiding values from the audit log.

  > **Warning.** Once enabled, this parameter can’t be disabled.

  Default: `false`

- **settings.apiserver.loadBalancer**
  object

  If set, a service `kube-system/d8-control-plane-apiserver` of the `LoadBalancer` type will be created.

  - **settings.apiserver.loadBalancer.annotations**
    object

    Annotations to attach to a service to fine-tune the load balancer.

    > **Caution!** The module does not take into account the specifics of setting annotations in various cloud environments. If the annotations for load balancer provisioning are only applied when creating a service, you will need to delete and add the `apiserver.loadBalancer` parameter to update such parameters.

  - **settings.apiserver.loadBalancer.port**
    integer

    External LoadBalancer TCP port.

    Default: `443`

    Allowed values: `1 <= X <= 65534`

  - **settings.apiserver.loadBalancer.sourceRanges**
    array of strings

    A list of CIDRs that are allowed to connect to the API.

    The cloud provider may not support this option or ignore it.

    - **Element of the array**
      string

      Pattern: `^[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\/[0-9]+$`

- **settings.apiserver.serviceAccount**
  object

  ServiceAccount issuing settings.

  Examples:

  ```
  serviceAccount: {}
  ```

  ```
  serviceAccount:
    additionalAPIAudiences:
    - istio-ca
  ```

  - **settings.apiserver.serviceAccount.additionalAPIAudiences**
    array of strings

    A list of API audiences to add when provisioning ServiceAccount tokens.

    The default audiences list includes:

- `serviceAccount.issuer`, if specified.
- `serviceAccount.additionalAPIIssuers`, if specified.
- `https://kubernetes.default.svc.${clusterDomain}`, for example, `https://kubernetes.default.svc.cluster.local` – this audience is always added for backward compatibility.

The service account token authenticator will validate that tokens used against the API are bound to at least one of these audiences.

This option is beneficial when migrating from one API issuer to another.

- **settings.apiserver.serviceAccount.additionalAPIIssuers**
  array of strings

  A list of additional issuers to include when provisioning ServiceAccount tokens.

  Issuers (`iss`) are used to verify the source of the tokens, ensuring they originate from trusted entities.

  The first issuer is automatically generated based on the template `https://kubernetes.default.svc.${clusterDomain}`, for example, `https://kubernetes.default.svc.cluster.local`. When multiple issuers are specified, the first issuer is used to generate tokens, and all provided issuers are accepted for token verification.

  This option is beneficial when migrating from one API issuer to another.

- **settings.apiserver.serviceAccount.issuer**
  string

  ServiceAccount issuer. This is the URL of the API server. The values of this field are used as the `iss` claim of the token and to verify Service Account JWT tokens.

  Note, all pods in the cluster using ServiceAccount tokens must be restarted upon changing this option.

  Example:

```
issuer: https://api.example.com
```

- **settings.enabledFeatureGates**
  array of strings

  List of feature gates enabled in the control plane.

  More information about feature gates is available in the Kubernetes documentation. The list of feature gates supported in DKP can be found on the module overview page.

  Default: `[]`

  - **Element of the array**
    string

    Pattern: `^[a-zA-Z]+$`

- **settings.etcd**
  object

  `etcd` parameters.

  - **settings.etcd.backup**
    object

    etcd backup parameters.

    - **settings.etcd.backup.cronSchedule**
      string

      Backup schedule etcd in cron format. The local time zone of the `kube-controller-manager` is used.

      Default: `0 0 * * *`

      Example:

```
cronSchedule: 0 1 * * *
```

    - **settings.etcd.backup.enabled**
      boolean

      Enable etcd backup.

      Default: `true`

        - **settings.etcd.backup.hostPath**
          string

          Host path on masters nodes for etcd backup.

          Default: `/var/lib/etcd`

      - **settings.etcd.maxDbSize**
        number

        quota-backend-bytes parameter. Deckhouse automatically manages the `quota-backend-bytes` parameter. If the `maxDbSize` parameter is set, deckhouse will use this value for the `quota-backend-bytes` etcd parameter.

        Minimum: 512MB.

        Maximum: 8GB.

        Experimental. It can be removed in the future.

        Allowed values: `536870912 <= X <= 8589934592`

  - **settings.failedNodePodEvictionTimeoutSeconds**
    integer

    The number of seconds after which pods will be deleted from the node with the `Unreachable` status.

    > **Note!** If you change the parameter, the pods must be restarted.

    Default: `300`

  - **settings.nodeMonitorGracePeriodSeconds**
    integer

    The number of seconds after which the node will enter the `Unreachable` status in case of lost connection.

    Default: `40`

  - **settings.rootKubeconfigSymlink**
    boolean

    Whether to maintain a symlink from `/root/.kube/config` to `/etc/kubernetes/admin.conf` on control-plane nodes.

    When the user-authz module is enabled and this parameter is `false`, the symlink is removed and the `NODE_ADMIN_KUBECONFIG` environment variable is set on control-plane-manager pods accordingly. Use `d8 k` with `--kubeconfig=/etc/kubernetes/admin.conf` (or other credentials) when you need API access from the node.

    When the `user-authz` module is disabled, `control-plane-manager` ignores this parameter and keeps the default (symlink present).

    Setting the value back to `true` restores the symlink on the next reconciliation.

    Default: `true`

    Examples:

    ```
    rootKubeconfigSymlink: true
    ```

    ```
    rootKubeconfigSymlink: false
    ```

Custom Resources

> The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

KubeSchedulerWebhookConfiguration

Scope: Cluster
Version: v1alpha1

Defines the configuration for connecting a third-party kube-scheduler extender webhook.

An external kube-scheduler webhook allows expanding the capabilities of the scheduler and consider more complex conditions when planning the load in the Kubernetes cluster.

- **webhooks**
  array of objects

  Required value

  - **webhooks.clientConfig**
    object

    Required value

    ClientConfig defines how to communicate with the webhook.

    - **webhooks.clientConfig.caBundle**
      string

      Required value

      caBundle is a PEM encoded CA bundle which will be used to validate the webhook’s server certificate.

      Pattern: `^(?:[A-Za-z0-9+\/]{4})*(?:[A-Za-z0-9+\/]{4}|[A-Za-z0-9+\/]{3}=|[A-Za-z0-9+\/]{2}={2})$`

    - **webhooks.clientConfig.service**
      object

      Required value

      Access to webhook via cluster service.

      - **webhooks.clientConfig.service.name**
        string

        Required value

        Service name.

        Pattern: `^[0-9a-z\-]+$`

        Length: `1..255`

      - **webhooks.clientConfig.service.namespace**
        string

        Required value

        Service namespace.

        Pattern: `^[0-9a-z\-]+$`

        Length: `1..255`

      - **webhooks.clientConfig.service.path**
        string

        Required value

        URI path.

        Pattern: `^[0-9a-zA-Z._\/-]+$`

        Length: `1..255`

      - **webhooks.clientConfig.service.port**
        integer

        Required value

        Service port.

        Allowed values: `1 <= X <= 65535`

  - **webhooks.failurePolicy**
    string

    Specifies scheduling should fail or not fail when the extender returns an error or is not reachable.

    Default: `Fail`

    Allowed values: `Fail`, `Ignore`

  - **webhooks.filterVerb**

string

Verb appended to `URLPrefix` when issuing a `filter` call to the scheduler extender. If omitted, defaults to `filter` (for backward compatibility, avoid making changes unless necessary). Set to an empty string `""` to disable `filter` calls for this extender.

- **webhooks.preemptVerb**
  string

  Verb appended to `URLPrefix` when issuing a `preempt` call to this extender. Leave empty (`""`) to disable `preempt` calls for this extender.

- **webhooks.prioritizeVerb**
  string

  Verb appended to `URLPrefix` when issuing a `prioritize` call to this extender. If omitted, defaults to `prioritize` (for backward compatibility, avoid making changes unless necessary). Set to an empty string `""` to disable `prioritize` calls for this extender.

- **webhooks.timeoutSeconds**
  integer

  Webhook timeout in seconds.

  Default: `10`

  Allowed values: `1 <= X <= 60`

- **webhooks.weight**
  integer

  Required value

  The numeric multiplier for the node scores that the prioritize call generates.

  Allowed values: `1 <= X`

FAQ

> **i** The module lifecycle stage: General Availability

How do I add a master node to a static or hybrid cluster?

> **i** It is important to have an odd number of masters to ensure a quorum.

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

When installing Deckhouse Kubernetes Platform with default settings, the NodeGroup `master` lacks the section `spec.staticInstances.labelSelector` with label filter settings for `staticInstances` resources. Because of this, after changing the number of `staticInstances` nodes in the NodeGroup `master` (parameter `spec.staticInstances.count`), when adding a regular node using Cluster API Provider Static (CAPS), it can be "intercepted" and added to the NodeGroup `master`, even if the corresponding `StaticInstance` (in `metadata`) specifies a label with a `role` different from `master`. To avoid this "interception", after installing DKP, edit the NodeGroup `master` — add the section `spec.staticInstances.labelSelector` with label filter settings for `staticInstances` resources. Example of NodeGroup `master` with `spec.staticInstances.labelSelector`:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: master
spec:
  nodeType: Static
  staticInstances:
    count: 2
    labelSelector:
      matchLabels:
        role: master
```

Next, when adding master nodes to the cluster using CAPS, specify the label specified in `spec.staticInstances.labelSelector` NodeGroup `master` in the corresponding `StaticInstance`. Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-master-1
  labels:
    # The label specified in spec.staticInstances.labelSelector NodeGroup master.
    role: master
spec:
  # Specify the IP address of the static node server.
  address: "<SERVER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
```

> **i** When adding new master nodes using CAPS and changing the number of master nodes in the NodeGroup `master` (parameter `spec.staticInstances.count`), please note the following:
>
> During the initial cluster installation, the configuration specifies the first master node on which the installation takes place. If, after the initial cluster installation, you need to create a multi-master cluster and add master nodes using CAPS, you must specify the number of nodes in the `spec.staticInstances.count` parameter of the NodeGroup `master` as one less than the desired number.
>
> For example, if you need to create a multi-master with three master nodes in `spec.staticInstances.count` NodeGroup `master`, specify the value `2` and create two `staticInstances` for the nodes to be added. After adding them to the cluster, the number of master nodes will be three: the master node on which the installation took place and two master nodes added using CAPS.

Otherwise, adding a master node to a static or hybrid cluster is similar to adding a regular node. To do this, use the corresponding examples. All the necessary actions to configure cluster control plane components on the new master nodes are performed automatically. Wait until the master nodes appear in `Ready` status.

How do I add master nodes to a cloud cluster?

The following describes the conversion of a single-master cluster into a multi-master.

> **i** Before adding nodes, ensure you have the required quotas in the cloud provider.
>
> It is important to have an odd number of masters to ensure a quorum.

> **⚠** If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

1. Make a backup of `etcd` and the `/etc/kubernetes` directory.
2. Transfer the archive to a server outside the cluster (e.g., on a local machine).
3. Ensure there are no alerts in the cluster that can prevent the creation of new master nodes.
4. Make sure that Deckhouse queue is empty:

   ```
   d8 system queue list
   ```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

   ```
   docker login registry.deckhouse.io
   ```

   During the login process, you will need to enter your `Username` and `Password`.

   > **i** When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Kubernetes Platform license key.

6. Run the appropriate edition and version of the Deckhouse installer container **on the local machine** (change the container registry address if necessary):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]'
'[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
  registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. **In the installer container**, run the following command to check the state before working:

   ```
   dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --
   ssh-host <MASTER-NODE-0-HOST>
   ```

   The command output should indicate that Terraform found no inconsistencies and no changes are required.

8. **In the installer container**, run the following command and specify the required number of replicas using the `masterNodeGroup.replicas` parameter:

   ```
   dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME>
   --ssh-user=<USERNAME> \
     --ssh-host <MASTER-NODE-0-HOST>
   ```

   > **i**
   >
   > For **Yandex Cloud**, when using external addresses on master nodes, the number of array elements in the masterNodeGroup.instanceClass.externalIPAddresses parameter must equal the number of master nodes. If `Auto` is used (public IP addresses are provisioned automatically), the number of array elements must still equal the number of master nodes.
   >
   > To illustrate, with three master nodes (`masterNodeGroup.replicas: 3`) and automatic address reservation, the `masterNodeGroup.instanceClass.externalIPAddresses` parameter would look as follows:
   >
   > ```
   > externalIPAddresses:
   > - "Auto"
   > - "Auto"
   > - "Auto"
   > ```

9. **In the installer container**, run the following command to start scaling:

   ```
   dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-
   host <MASTER-NODE-0-HOST>
   ```

10. Wait until the required number of master nodes are `Ready` and all `control-plane-manager` instances are up and running:

    ```
    d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady -l app=d8-control-plane-
    manager
    ```

How do I reduce the number of master nodes in a cloud cluster?

The following describes the conversion of a multi-master cluster into a single-master.

> ⚠ The steps described below must be performed from the first in order of the master node of the cluster (master-0). This is because the cluster is always scaled in order: for example, it is impossible to delete nodes master-0 and master-1, leaving master-2.

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

1. Create a backup of etcd and the `/etc/kubernetes` directory.
2. Copy the resulting archive outside the cluster (e.g., to a local machine).
3. Ensure there are no alerts in the cluster that may interfere with the master node update process.
4. Make sure the DKP queue is empty:

```
d8 system queue list
```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

   ```
   docker login registry.deckhouse.io
   ```

   During the login process, you will need to enter your `Username` and `Password`.

   > **i** When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Kubernetes Platform license key.

6. On the **local machine**, run the DKP installer container for the corresponding edition and version (change the container registry address if needed):

   ```
   DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
   DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]'
   '[:lower:]' )
   docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
     registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
   ```

7. **In the installer container**, run the following command and set `masterNodeGroup.replicas` to `1`:

   ```
   dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME>
   \
     --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
   ```

   > **i** For **Yandex Cloud**, if external IPs are used for master nodes, the number of items in the `masterNodeGroup.instanceClass.externalIPAddresses` array must match the number of master nodes. Even when using `Auto` (automatic public IP allocation), the number of entries must still match.
   >
   > For example, for a single master node (`masterNodeGroup.replicas: 1`) and automatic IP assignment, the `masterNodeGroup.instanceClass.externalIPAddresses` section would look like:
   >
   > ```
   > externalIPAddresses:
   > - "Auto"
   > ```

8. **In the installer container**, run the following command to trigger the scaling operation:

   ```
   dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-
   host <MASTER-NODE-0-HOST>
   ```

   > **i** For **OpenStack** and **VK Cloud (OpenStack)**, after confirming the node deletion, it is extremely important to check the disk deletion `<prefix>kubernetes-data-N` in Openstack itself.
   >
   > For example, when deleting the `cloud-demo-master-2` node in the Openstack web interface or in the OpenStack CLI, it is necessary to check the absence of the `cloud-demo-kubernetes-data-2` disk.
   >
   > If the kubernetes-data disk remains, there may be problems with ETCD operation as the number of master nodes increases.

9. Check the Deckhouse queue and make sure that there are no errors with the command:

   ```
   d8 system queue list
   ```

How do I dismiss the master role while keeping the node?

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module’s data before making any changes.

1. Make a backup of `etcd` and the `/etc/kubernetes` directory.
2. Transfer the archive to a server outside the cluster (e.g., on a local machine).
3. Ensure there are no alerts in the cluster that can prevent the update of the master nodes.
4. Make sure that Deckhouse queue is empty:

   ```
   d8 system queue list
   ```

5. Remove the following labels:
   - `node-role.kubernetes.io/control-plane`
   - `node-role.kubernetes.io/master`
   - `node.deckhouse.io/group`

   Use the following command to remove labels:

   ```
   d8 k label node <MASTER-NODE-N-NAME> node-role.kubernetes.io/control-plane- node-role.kubernetes.io/master-
   node.deckhouse.io/group-
   ```

6. Make sure that the master node to be deleted is no longer listed as a member of the etcd cluster:

   ```
   for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
     d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
     --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
     --endpoints https://127.0.0.1:2379/ member list -w table
     if [ $? -eq 0 ]; then
       break
     fi
   done
   ```

7. Exec to the node and run the following commands:

   ```
   rm -f /etc/kubernetes/manifests/{etcd,kube-apiserver,kube-scheduler,kube-controller-manager}.yaml
   rm -f /etc/kubernetes/{scheduler,controller-manager}.conf
   rm -f /etc/kubernetes/authorization-webhook-config.yaml
   rm -f /etc/kubernetes/admin.conf /root/.kube/config
   rm -rf /etc/kubernetes/deckhouse
   rm -rf /etc/kubernetes/pki/{ca.key,apiserver*,etcd/,front-proxy*,sa.*}
   rm -rf /var/lib/etcd/member/
   ```

How do I switch to a different OS image in a multi-master cluster?

The OS change method depends on the cluster type: in a cloud cluster, nodes are replaced with `dhctl converge`; in a static cluster, nodes are replaced manually, one at a time.

In a cloud cluster

To change the OS image in a cloud cluster with multiple master nodes, follow these steps:

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

1. Make a backup of `etcd` and the `/etc/kubernetes` directory.
2. Transfer the archive to a server outside the cluster (e.g., on a local machine).
3. Ensure there are no alerts in the cluster that can prevent the update of the master nodes.
4. Make sure that Deckhouse queue is empty:

   ```
   d8 system queue list
   ```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

   ```
   docker login registry.deckhouse.io
   ```

   During the login process, you will need to enter your `Username` and `Password`.

> **i** When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Kubernetes Platform license key.

6. Run the appropriate edition and version of the Deckhouse installer container **on the local machine** (change the container registry address if necessary):

```
DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]'
'[:lower:]' )
docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
  registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
```

7. **In the installer container**, run the following command to check the state before working:

```
dhctl terraform check --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
  --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

   The command output should indicate that Terraform found no inconsistencies and no changes are required.

8. **In the installer container**, run the following command and specify the required OS image using the `masterNodeGroup.instanceClass` parameter (specify the addresses of all master nodes using the `--ssh-host` parameter):

```
dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME>
--ssh-user=<USERNAME> \
  --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

9. **In the container with the installer**, run the following command to update the nodes:

   Carefully review the actions that converge plans to perform when it asks for confirmation.

   When the command is executed, the nodes will be replaced by new nodes with confirmation on each node. The replacement will be performed one by one in reverse order (2,1,0).

```
dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> \
  --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST> --ssh-host <MASTER-NODE-2-HOST>
```

   The following steps should be performed **sequentially on each** master node, starting with the node with the highest number (suffix 2) and ending with the node with the lowest number (suffix 0).

10. **On the newly created node**, check the systemd-unit log for the `bashible.service`. Wait until the node configuration is complete (you will see a message `nothing to do` in the log):

```
journalctl -fu bashible.service
```

11. Make sure the node is listed as an etcd cluster member:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

12. Make sure `control-plane-manager` is running on the node:

```
d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
  -l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER-NODE-N-NAME>
```

13. Proceed to update the next node (repeat the steps above).

In a static cluster

The following instruction describes OS replacement on master nodes that were added to the cluster **manually** using the `bootstrap.sh` script. Perform the steps for each master node, one node at a time. Do not proceed to the next node until the current node has rejoined the cluster and is healthy.

> ⚠ If master nodes are managed by Cluster API Provider Static (CAPS) through StaticInstance resources, do not use this instruction. First delete the StaticInstance, install the required OS, and add the node to the `master` NodeGroup again.

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

To change the OS of a manually added master node, follow these steps:

1. Make a backup of `etcd` and the `/etc/kubernetes` directory. If the `stronghold` module is enabled, make sure its data has also been backed up.
2. Check the cluster health and make sure there are no alerts or pending Deckhouse queue tasks:

   ```
   d8 status
   ```

3. Remove the master node labels from the node:

   ```
   d8 k label node <MASTER_NODE_NAME> node-role.kubernetes.io/control-plane- \
     node-role.kubernetes.io/master- node.deckhouse.io/group-
   ```

   where `<MASTER_NODE_NAME>`: Name of the master node being changed.
4. Make sure the node has been removed from the etcd cluster member list:

   ```
   for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
     d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
       --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
       --endpoints https://127.0.0.1:2379/ member list -w table
     if [ $? -eq 0 ]; then
       break
     fi
   done
   ```

5. Evict workloads from the node with `d8 k drain`:

   ```
   d8 k drain <MASTER_NODE_NAME> --ignore-daemonsets --delete-emptydir-data
   ```

6. Force-delete any Pods remaining on the node:

   ```
   d8 k delete pods --all-namespaces --field-selector spec.nodeName=<MASTER_NODE_NAME> --force
   ```

7. Delete the Node object:

   ```
   d8 k delete node <MASTER_NODE_NAME>
   ```

8. Clean up the DKP data on the removed master node. This command irreversibly removes Kubernetes and DKP data from the node. Before running it, make sure you have selected the correct node and created the required backups:

   ```
   bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
   ```

9. Install the required OS on the node.
10. On one of the remaining master nodes in the cluster, retrieve and decode the script for adding a master node:

    ```
    d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-master \
      -o jsonpath='{.data.bootstrap\.sh}' | base64 -d > bootstrap.sh
    ```

11. Securely copy the `bootstrap.sh` file obtained in the previous step to the node being added and run it as `root` on that node:

```
bash bootstrap.sh
```

12. **On the node being added**, check the systemd-unit log for the `bashible.service`. Wait until the node configuration is complete (you will see a message `nothing to do` in the log):

    ```
    journalctl -fu bashible.service
    ```

13. Wait until the node reaches the `Ready` status:

    ```
    d8 k wait node <MASTER_NODE_NAME> --for=condition=Ready --timeout=10m
    ```

14. Make sure the node is listed as an etcd cluster member:

    ```
    for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
      d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
       --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
       --endpoints https://127.0.0.1:2379/ member list -w table
      if [ $? -eq 0 ]; then
        break
      fi
    done
    ```

15. Make sure `control-plane-manager` is running on the node:

    ```
    d8 k -n kube-system wait pod --timeout=10m --for=condition=ContainersReady \
      -l app=d8-control-plane-manager --field-selector spec.nodeName=<MASTER_NODE_NAME>
    ```

16. Make sure there are no alerts or pending queue tasks:

    ```
    d8 status
    ```

17. Repeat the procedure for the next master node.

How do I switch to a different OS image in a single-master cluster?

The method depends on the cluster type: first add extra master nodes, replace the OS in multi-master mode, then restore the original number of master nodes.

> ⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before changing master nodes. Create a backup of the module's data before making any changes.

In a cloud cluster

1. Convert your single-master cluster to a multi-master one, as described in the guide on adding master nodes to a cluster.
2. Change the OS on the master nodes according to the instructions.
3. Convert your multi-master cluster to a single-master one according to the guide on excluding master nodes from the cluster.

In a static cluster

1. Add extra master nodes according to the instructions.
2. Change the OS on the master nodes according to the instructions.
3. Remove extra master nodes from the control plane role according to the instructions. Then delete them from the cluster with `d8 k delete node <MASTER_NODE_NAME>` and power off the corresponding servers.

How to configure HA mode with two master nodes and an arbiter node?

Deckhouse Kubernetes Platform allows you to configure HA mode with two master nodes and an arbiter node. This approach allows you to meet HA requirements in conditions of limited resources.

Only etcd is placed on the arbiter node, without the other control plane components. This node is used to ensure the etcd quorum.

Requirements for the arbiter node:

- At least 2 CPU cores
- At least 4 GB of RAM
- At least 8 GB of disk space for etcd

The network latency requirements for the arbiter node are similar to those for the master nodes.

Configuring in a cloud cluster

The example below applies to a cloud cluster with three master nodes. To configure HA mode with two master nodes and an arbiter node in a cloud cluster, you need to remove one master node from the cluster and add one arbiter node.

To do this, follow these steps:

⚠ The following steps must be performed starting from the first master node (`master-0`) in the cluster. This is because the cluster scales in order — for example, it is not possible to remove `master-0` and `master-1` while leaving `master-2`.

⚠ If your cluster uses the `stronghold` module, make sure the module is fully operational before adding or removing a master node. Create a backup of the module's data before making any changes.

1. Create a backup of etcd and the `/etc/kubernetes` directory.
2. Copy the resulting archive outside the cluster (for example, to a local machine).
3. Ensure there are no alerts in the cluster that may interfere with the master node update process.
4. Make sure the DKP queue is empty:

   ```
   d8 system queue list
   ```

5. **On your local machine**, log in to the registry (change the container registry address if necessary):

   ```
   docker login registry.deckhouse.io
   ```

   During the login process, you will need to enter your `Username` and `Password`.

   i When logging in to the `registry.deckhouse.io` registry, the `Username` field must be set to `license-token`, and the `Password` field must contain the Deckhouse Kubernetes Platform license key.

6. On the **local machine**, run the DKP installer container for the corresponding edition and version (change the container registry address if needed):

   ```
   DH_VERSION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/version}')
   DH_EDITION=$(d8 k -n d8-system get deployment deckhouse -o
   jsonpath='{.metadata.annotations.core\.deckhouse\.io\/edition}' | tr '[:upper:]'
   '[:lower:]' )
   docker run --pull=always -it -v "$HOME/.ssh/:/tmp/.ssh/" \
     registry.deckhouse.io/deckhouse/${DH_EDITION}/install:${DH_VERSION} bash
   ```

7. **In the installer container**, run the following command:

   ```
   dhctl config edit provider-cluster-configuration --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME>
   \
     --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST>
   ```

   Change the cloud provider settings:

   - Set `masterNodeGroup.replicas` to `2`.
   - Create a NodeGroup for the arbiter node. The arbiter node **must have** the label `node.deckhouse.io/etcd-arbiter: ""` and a taint that prevents user workloads from being placed on it. Example of a NodeGroup description for the arbiter node:

```
nodeGroups:
  - name: arbiter
    replicas: 1
    nodeTemplate:
      labels:
        node.deckhouse.io/etcd-arbiter: ""
      taints:
        - key: node.deckhouse.io/etcd-arbiter
          effect: NoSchedule
    zones:
      - europe-west3-b
    instanceClass:
      machineType: n1-standard-4
  # ... the rest of the manifest
```

- Save your changes.

> **i** For **Yandex Cloud**, if external IPs are used for master nodes, the number of items in the `masterNodeGroup.instanceClass.externalIPAddresses` array must match the number of master nodes. Even when using `Auto` (automatic public IP allocation), the number of entries must still match.
>
> For example, for a single master node (`masterNodeGroup.replicas: 1`) and automatic IP assignment, the `masterNodeGroup.instanceClass.externalIPAddresses` section would look as follows:
>
> ```
> externalIPAddresses:
> - "Auto"
> ```

8. **In the installer container**, run the following command to trigger the scaling operation:

   ```
   dhctl converge --ssh-agent-private-keys=/tmp/.ssh/<SSH_KEY_FILENAME> --ssh-user=<USERNAME> --ssh-host <MASTER-NODE-0-HOST> --ssh-host <MASTER-NODE-1-HOST>
   ```

   > **i** **Important**. For **OpenStack** and **VK Cloud (OpenStack)**, after confirming the node deletion, it is extremely important to check the disk deletion `<prefix>kubernetes-data-N` in OpenStack itself.
   >
   > For example, when deleting the `cloud-demo-master-2` node in the OpenStack web interface or in the OpenStack CLI, it is necessary to check the absence of the `cloud-demo-kubernetes-data-2` disk.
   >
   > If the `kubernetes-data` disk remains, there may be problems with etcd operation as the number of master nodes increases.

9. Check the Deckhouse queue with the following command and make sure there are no errors:

   ```
   d8 system queue list
   ```

Configuring in a static cluster

To configure HA mode with two master nodes and an arbiter node in a static cluster, follow these steps:

1. Create a NodeGroup for the arbiter node. The arbiter node **must have** the label `node.deckhouse.io/etcd-arbiter: “”` and a taint that prevents user workloads from being placed on it. Example of a NodeGroup description for the arbiter node:

```
apiVersion: deckhouse.io/v1
  kind: NodeGroup
  metadata:
    name: arbiter
  spec:
    nodeType: Static
    nodeTemplate:
      labels:
        node.deckhouse.io/etcd-arbiter: ""
      taints:
        - key: node.deckhouse.io/etcd-arbiter
          effect: NoSchedule
  # ... the rest of the manifest
```

2. Add a node to the cluster that will be used as an arbiter node in a way that is convenient for you.
3. Ensure that the added arbiter node is in the list of etcd cluster members.
4. Remove one master node from the cluster.

How do I view the list of etcd members?

Option 1

Use the `etcdctl member list` command.

Example:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ member list -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

Warning. The last parameter in the output table shows etcd member is in `learner` state, is not in `leader` state.

Option 2

To obtain information about etcd cluster nodes in tabular form, use the `etcdctl endpoint status` command. For the leader, the `IS LEADER` column will show `true`.

Example:

```
for pod in $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name); do
  d8 k -n kube-system exec "$pod" -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ endpoint status --cluster -w table
  if [ $? -eq 0 ]; then
    break
  fi
done
```

What if something went wrong?

During its operation, `control-plane-manager` automatically creates backups of configurations and data that may be useful in case of issues. These backups are stored in the `/etc/kubernetes/deckhouse/backup` directory. If errors or unforeseen situations occur during operation, you can use these backups to restore to the previous stable state.

What if the etcd cluster fails?

If the etcd cluster is not functioning and it cannot be restored from a backup, you can attempt to rebuild it from scratch by following the steps below.

1. First, on all nodes that are part of your etcd cluster, except for one, remove the `etcd.yaml` manifest located in the `/etc/kubernetes/manifests/` directory. This last node will serve as a starting point for the new multi-master cluster.
2. On the last node, edit etcd manifest `/etc/kubernetes/manifests/etcd.yaml` and add the parameter `--`

`force-new-cluster` to `spec.containers.command`.
3. After the new cluster is ready, remove the `--force-new-cluster` parameter.

> ⚠ This operation is unsafe and breaks the guarantees given by the consensus protocol. Note that it brings the cluster to the state that was saved on the node. Any pending entries will be lost.

What if etcd restarts with an error?

This method may be necessary if the `--force-new-cluster` option doesn't restore etcd work. Such a scenario can occur during an unsuccessful converge of master nodes, where a new master node was created with an old etcd disk, changed its internal address, and other master nodes are absent. Symptoms indicating the need for this method include: the etcd container being stuck in an endless restart with the log showing the error: `panic: unexpected removal of unknown remote peer`.

1. Find the `etcdutl` utility on the master node and copy the executable to `/usr/local/bin/`:

   ```
   cp $(find /var/lib/containerd/ \
   -name etcdutl -print -quit) /usr/local/bin/etcdutl
   ```

2. Create a new etcd database snapshot from the current local snapshot (`/var/lib/etcd/member/snap/db`):

   ```
   etcdutl snapshot restore /var/lib/etcd/member/snap/db --name <HOSTNAME> \
   --initial-cluster=<HOSTNAME>=https://<ADDRESS>:2380 --initial-advertise-peer-
   urls=https://<ADDRESS>:2380 \
   --skip-hash-check=true --data-dir /var/lib/etcdtest
   ```

   where:

   - `<HOSTNAME>`: Name of the master node.
   - `<ADDRESS>`: Address of the master node.

3. Execute the following commands to use the new snapshot:

   ```
   cp -r /var/lib/etcd /tmp/etcd-backup
   rm -rf /var/lib/etcd
   mv /var/lib/etcdtest /var/lib/etcd
   ```

4. Locate the `etcd` and `api-server` containers:

   ```
   crictl ps -a | egrep "etcd|apiserver"
   ```

5. Remove the `etcd` and `api-server` containers:

   ```
   crictl rm <CONTAINER-ID>
   ```

6. Restart the master node.

What to do if the database volume of etcd reaches the limit set in quota-backend-bytes?

When the database volume of etcd reaches the limit set by the `quota-backend-bytes` parameter, it switches to "read-only" mode. This means that the etcd database stops accepting new entries but remains available for reading data. You can tell that you are facing a similar situation by executing the command:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name |
sed -n 1p) -- \
  etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt \
  --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ \
  endpoint status -w table --cluster
```

If you see a message like `alarm:NOSPACE` in the `ERRORS` field, you need to take the following steps:

1. On **each master node** (from the list of the etcd cluster members obtained earlier), find the line with `--quota-backend-bytes` in the etcd pod manifest at `/etc/kubernetes/manifests/etcd.yaml` and double the value. If there is no such line, add one, for example: `- --quota-backend-bytes=8589934592`. This sets the limit to 8 GB.

2. Disarm the active alarm that occurred due to reaching the limit. To do this, execute the command:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name
| sed -n 1p) -- \
etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt \
--cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key \
--endpoints https://127.0.0.1:2379/ alarm disarm
```

3. Change the maxDbSize parameter in the `control-plane-manager` settings to match the value specified in the manifest.

How to compact etcd storage

> ⚠ Before compacting etcd storage, back up etcd.

To view the size of the etcd database on a specific node before and after compaction, use the command (where `NODE_NAME` is the name of the master node):

```
d8 k -n kube-system exec -it etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  endpoint status --cluster -w table
```

Output example (the size of the etcd database on the node is specified in the `DB SIZE` column):

```
+----------------------------+------------------+---------+-----------------+---------+----
----+----------------------+--------+------------+------------+-----------+------------+---
-----------------+--------+--------------------------+-------------------+
|          ENDPOINT          |        ID        | VERSION | STORAGE VERSION | DB SIZE | IN
USE | PERCENTAGE NOT IN USE | QUOTA  | IS LEADER  | IS LEARNER | RAFT TERM | RAFT INDEX |
RAFT APPLIED INDEX | ERRORS | DOWNGRADE TARGET VERSION | DOWNGRADE ENABLED |
+----------------------------+------------------+---------+-----------------+---------+----
----+----------------------+--------+------------+------------+-----------+------------+---
-----------------+--------+--------------------------+-------------------+
| https://192.168.199.80:2379 | 489a8af1e7acd7a0 |   3.6.1 |           3.6.0 |   76 MB |  62
MB |                  20% | 2.1 GB |       true |      false |        56 |  258054684 |
258054684 |        |                          |            false |
+----------------------------+------------------+---------+-----------------+---------+----
----+----------------------+--------+------------+------------+-----------+------------+---
-----------------+--------+--------------------------+-------------------+
| https://192.168.199.81:2379 | 589a8ad1e7ccd7b0 |   3.6.1 |           3.6.0 |   76 MB |  62
MB |                  20% | 2.1 GB |      false |      false |        56 |  258054685 |
258054685 |        |                          |            false |
+----------------------------+------------------+---------+-----------------+---------+----
----+----------------------+--------+------------+------------+-----------+------------+---
-----------------+--------+--------------------------+-------------------+
| https://192.168.199.82:2379 | 229a8cd1e7bcd7a0 |   3.6.1 |           3.6.0 |   76 MB |  62
MB |                  20% | 2.1 GB |      false |      false |        56 |  258054685 |
258054685 |        |                          |            false |
+----------------------------+------------------+---------+-----------------+---------+----
----+----------------------+--------+------------+------------+-----------+------------+---
-----------------+--------+--------------------------+-------------------+
```

How to compact etcd storage on a node in a single-master cluster

> ⚠ Compacting etcd storage is a resource-intensive operation that temporarily blocks etcd from running on that node. Keep this in mind when choosing a time to perform the operation in a cluster with a single master node.

To compact etcd storage in a cluster with a single master node, use the following command (where `NODE_NAME` is the name of the master node):

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt \
  --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

Example output when the operation is successful:

```
Finished defragmenting etcd member[https://localhost:2379]. took 848.948927ms
```

> **i** If a timeout error occurs, increase the value of the `—command-timeout` parameter from the command above until compaction completes successfully.

How to compact etcd storage in a cluster with multiple master nodes

To compact etcd storage in a cluster with multiple master nodes:

1. Get a list of etcd pods. To do this, use the following command:

   ```
   d8 k -n kube-system get pod -l component=etcd -o wide
   ```

   Example output:

   ```
   NAME           READY    STATUS    RESTARTS   AGE     IP              NODE
   NOMINATED NODE   READINESS GATES
   etcd-master-0   1/1     Running   0          3d21h   192.168.199.80  master-0    <none>
   <none>
   etcd-master-1   1/1     Running   0          3d21h   192.168.199.81  master-1    <none>
   <none>
   etcd-master-2   1/1     Running   0          3d21h   192.168.199.82  master-2    <none>
   <none>
   ```

2. Identify the leader master node. To do this, contact any etcd pod and get a list of nodes participating in the etcd cluster using the command (where `NODE_NAME` is the name of the master node):

   ```
   d8 k -n kube-system exec -it etcd-NODE_NAME -- /usr/bin/etcdctl \
     --cert=/etc/kubernetes/pki/etcd/server.crt \
     --key=/etc/kubernetes/pki/etcd/server.key \
     --cacert=/etc/kubernetes/pki/etcd/ca.crt \
     endpoint status --cluster -w table
   ```

   Output example (the leader in the `IS LEADER` column will have the value `true`):

```
+----------------------------+-----------------+---------+----------------+---------
+--------+----------------------+--------+------------+------------+-----------+------
------+--------------------+--------+-------------------------+------------------+
|          ENDPOINT          |        ID        | VERSION | STORAGE VERSION | DB SIZE
| IN USE | PERCENTAGE NOT IN USE | QUOTA  | IS LEADER  | IS LEARNER | RAFT TERM | RAFT
INDEX | RAFT APPLIED INDEX | ERRORS | DOWNGRADE TARGET VERSION | DOWNGRADE ENABLED |
+----------------------------+-----------------+---------+----------------+---------
+--------+----------------------+--------+------------+------------+-----------+------
------+--------------------+--------+-------------------------+------------------+
| https://192.168.199.80:2379 | 489a8af1e7acd7a0 |   3.6.1 |          3.6.0 |   76 MB
|  62 MB |                  20% | 2.1 GB |       true |      false |        56 |
258054684 |          258054684 |        |                         |             false
|
+----------------------------+-----------------+---------+----------------+---------
+--------+----------------------+--------+------------+------------+-----------+------
------+--------------------+--------+-------------------------+------------------+
| https://192.168.199.81:2379 | 589a8ad1e7ccd7b0 |   3.6.1 |          3.6.0 |   76 MB
|  62 MB |                  20% | 2.1 GB |      false |      false |        56 |
258054685 |          258054685 |        |                         |             false
|
+----------------------------+-----------------+---------+----------------+---------
+--------+----------------------+--------+------------+------------+-----------+------
------+--------------------+--------+-------------------------+------------------+
| https://192.168.199.82:2379 | 229a8cd1e7bcd7a0 |   3.6.1 |          3.6.0 |   76 MB
|  62 MB |                  20% | 2.1 GB |      false |      false |        56 |
258054685 |          258054685 |        |                         |             false
|
+----------------------------+-----------------+---------+----------------+---------
+--------+----------------------+--------+------------+------------+-----------+------
------+--------------------+--------+-------------------------+------------------+
```

3. Compact etcd storage on the etcd cluster member nodes one by one. Use the following command (where `NODE_NAME` is the name of the master node):

> **i** Important: compact the leader node last.
>
> Restoring etcd on a node after compaction may take some time. It is recommended to wait at least a minute before proceeding to compact etcd storage on the next node.

```
d8 k -n kube-system exec -ti etcd-NODE_NAME -- /usr/bin/etcdctl \
  --cacert /etc/kubernetes/pki/etcd/ca.crt \
  --cert /etc/kubernetes/pki/etcd/ca.crt \
  --key /etc/kubernetes/pki/etcd/ca.key \
  --endpoints https://127.0.0.1:2379/ defrag --command-timeout=30s
```

Example output when the operation is successful:

```
Finished defragmenting etcd member[https://localhost:2379]. took 848.948927ms
```

> **i** If a timeout error occurs, increase the value of the `–command-timeout` parameter from the command above until compaction completes successfully.

Cluster admin access model

The control-plane-manager module maintains several kubeconfig files on master nodes. Understanding their purpose is important for secure cluster administration.

Kubeconfig files on master nodes

File	Identity	Purpose

File	Identity	Purpose
`/etc/kubernetes/admin.conf`	`kubernetes-admin` (`kubeadm:cluster-admins` group)	Machine kubeconfig for kubeadm internals (join, renewal). The group is bound to the built-in wildcard `cluster-admin` role.
`/etc/kubernetes/super-admin.conf`	`kubernetes-super-admin` (`system:masters` group)	Break-glass emergency credential. Bypasses RBAC entirely. Restrict access to this file to trusted recovery scenarios.
`/etc/kubernetes/controller-manager.conf`	`system:kube-controller-manager`	Used by kube-controller-manager.
`/etc/kubernetes/scheduler.conf`	`system:kube-scheduler`	Used by kube-scheduler.

RBAC-based admin access

Starting from Kubernetes 1.29, kubeadm generates `admin.conf` with the `kubeadm:cluster-admins` group instead of `system:masters`. This provides RBAC-controlled admin access that can be revoked by removing the RBAC binding objects for `kubeadm:cluster-admins`.

Deckhouse binds the `kubeadm:cluster-admins` group to the built-in wildcard ClusterRole `cluster-admin` (the same effective model as a plain kubeadm cluster without extra RBAC), regardless of whether the user-authz module is enabled. For full unrestricted access that also bypasses RBAC, use `super-admin.conf`.

Recommended admin access

When the user-authn module is enabled, use personalized OIDC-based kubeconfig obtained through the kubeconfig generator. This provides individual accountability and audit trail.

When `user-authn` is disabled, administrators can explicitly use the admin kubeconfig on a master node:

```
d8 k --kubeconfig=/etc/kubernetes/admin.conf <command>
```

Root kubeconfig symlink

By default, the CPM module creates a symlink `/root/.kube/config` -> `/etc/kubernetes/admin.conf` on master nodes, allowing root to run `d8 k` without specifying `--kubeconfig`.

When the **user-authz** module is enabled, you can disable this symlink by setting `rootKubeconfigSymlink: false` in the **control-plane-manager** module configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    rootKubeconfigSymlink: false
```

If **user-authz** is disabled, CPM does not apply this setting and keeps the default symlink behavior.

When the symlink is disabled (and **user-authz** is enabled), the symlink is removed when it pointed to `admin.conf`. Use personalized credentials or pass `--kubeconfig` explicitly.

Security hardening

The CPM module automatically restricts file permissions on `admin.conf` and `super-admin.conf` to `0600` (owner read/write only) during every reconciliation cycle. This prevents unauthorized users from reading these sensitive credentials.

Break-glass access

In emergency situations (RBAC misconfiguration, webhook failures), use `super-admin.conf`:

```
d8 k --kubeconfig=/etc/kubernetes/super-admin.conf <command>
```

This credential bypasses all RBAC checks. Use it only as a last resort and restrict who can read the file.

How do I configure additional audit policies?

1. Enable the auditPolicyEnabled flag in the module configuration:

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: control-plane-manager
   spec:
     version: 2
     settings:
       apiserver:
         auditPolicyEnabled: true
   ```

2. Create the `kube-system/audit-policy` Secret containing a Base64 encoded YAML file:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: audit-policy
     namespace: kube-system
   data:
     audit-policy.yaml: <base64>
   ```

   The minimum viable example of the `audit-policy.yaml` file looks as follows:

   ```
   apiVersion: audit.k8s.io/v1
   kind: Policy
   rules:
   - level: Metadata
     omitStages:
     - RequestReceived
   ```

   You can find detailed information on how to configure `audit-policy.yaml` file here:

   - The official Kubernetes documentation;
   - The code of the generator script used in GCE.

How to omit Deckhouse built-in policy rules?

Set the apiserver.basicAuditPolicyEnabled module parameter to `false`.

An example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  settings:
    apiserver:
      auditPolicyEnabled: true
      basicAuditPolicyEnabled: false
```

How to stream audit log to stdout instead of files?

Set the apiserver.auditLog.output parameter to `Stdout`.

An example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  settings:
    apiserver:
      auditPolicyEnabled: true
      auditLog:
        output: Stdout
```

How to deal with the audit log?

There must be some `log scraper` on master nodes *(log-shipper, promtail, filebeat)* that will monitor the log file:

```
/var/log/kube-audit/audit.log
```

The following fixed parameters of log rotation are in use:

- The maximum disk space is limited to `1000 Mb`.
- Logs older than `30 days` will be deleted.

Depending on the `Policy` settings and the number of requests to the `apiserver`, the amount of logs collected may be high. Thus, in some cases, logs can only be kept for less than 30 minutes.

> ⚠ The current implementation of this feature isn't safe and may lead to a temporary failure of the control plane.
>
> The `apiserver` will not be able to start if there are unsupported options or a typo in the Secret.

If `apiserver` is unable to start, you have to manually disable the `--audit-log-*` parameters in the `/etc/kubernetes/manifests/kube-apiserver.yaml` manifest and restart apiserver using the following command:

```
docker stop $(docker ps | grep kube-apiserver- | awk '{print $1}')
# Or (depending on your CRI).
crictl stopp $(crictl pods --name=kube-apiserver -q)
```

After the restart, you will be able to fix the Secret or delete it:

```
d8 k -n kube-system delete secret audit-policy
```

How do I speed up the restart of Pods if the connection to the node has been lost?

By default, a node is marked as unavailable if it does not report its state for 40 seconds. After another 5 minutes, its Pods will be rescheduled to other nodes. Thus, the overall application unavailability lasts approximately 6 minutes.

In specific cases, if an application cannot run in multiple instances, there is a way to lower its unavailability time:

1. Reduce the period required for the node to become `Unreachable` if the connection to it is lost by setting the `nodeMonitorGracePeriodSeconds` parameter.
2. Set a lower timeout for evicting Pods on a failed node using the `failedNodePodEvictionTimeoutSeconds` parameter.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  settings:
    nodeMonitorGracePeriodSeconds: 10
    failedNodePodEvictionTimeoutSeconds: 50
```

In this case, if the connection to the node is lost, the applications will be restarted in about 1 minute.

Both these parameters directly impact the CPU and memory resources consumed by the control plane. By lowering timeouts, we force system components to send statuses more frequently and check the resource state more often.

When deciding on the appropriate threshold values, consider resources consumed by the control nodes (graphs can help you with this). Note that the lower parameters are, the more resources you may need to allocate to these nodes.

etcd backup and restore

What is done automatically

CronJob `kube-system/d8-etcd-backup-*` is automatically started at 00:00 UTC+0. The result is saved in `/var/lib/etcd/etcd-backup.tar.gz` on all nodes with `control-plane` in the cluster (master nodes).

How to manually backup etcd

Using Deckhouse CLI (Deckhouse Kubernetes Platform v1.65+)

Starting with Deckhouse Kubernetes Platform v1.65, a new `d8 backup etcd` tool is available for taking snapshots of etcd state.

```
d8 backup etcd ./etcd-backup.snapshot
```

Using bash (Deckhouse Kubernetes Platform v1.64 and older)

Login into any control-plane node with `root` user and use next script:

```
#!/usr/bin/env bash
set -e

pod=etcd-`hostname`
d8 k -n kube-system exec "$pod" -- /usr/bin/etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt --cert
/etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key --endpoints https://127.0.0.1:2379/
snapshot save /var/lib/etcd/${pod##*/}.snapshot && \
mv /var/lib/etcd/"${pod##*/}.snapshot" etcd-backup.snapshot && \
cp -r /etc/kubernetes/ ./ && \
tar -cvzf kube-backup.tar.gz ./etcd-backup.snapshot ./kubernetes/
rm -r ./kubernetes ./etcd-backup.snapshot
```

In the current directory etcd snapshot file `kube-backup.tar.gz` will be created from one of an etcd cluster members. From this file, you can restore the previous etcd cluster state in the future.

Also, we recommend making a backup of the `/etc/kubernetes` directory, which contains:

- manifests and configurations of control-plane components;
- Kubernetes cluster PKI.

This directory will help to quickly restore a cluster in case of complete loss of control-plane nodes without creating a new cluster and without rejoin the remaining nodes into the new cluster.

We recommend encrypting etcd snapshot backups as well as backup of the directory `/etc/kubernetes/` and saving them outside the Deckhouse cluster. You can use one of third-party files backup tools, for example: Restic, Borg, Duplicity, etc.

You can see documentation for learn about etcd disaster recovery procedures from snapshots.

How do perform a full recovery of the cluster state from an etcd backup?

The following steps will be described to restore to the previous state of the cluster from a backup in case of complete data loss.

Restoring a single-master cluster

Follow these steps to restore a single-master cluster on master node:

1. Find `etcdutl` utility on the master-node and copy the executable to `/usr/local/bin/`:

```
cp $(find /var/lib/containerd/ \
-name etcdutl -print -quit) /usr/local/bin/etcdutl
```

Check the version of `etcdutl` using the command:

```
etcdutl version
```

The result must be a correct output of `etcdutl version` command without errors.

Alternatively, you can download etcdutl executable to the node (preferably its version is the same as the etcd version in the cluster):

```
wget "https://github.com/etcd-io/etcd/releases/download/v3.6.1/etcd-v3.6.1-linux-amd64.tar.gz"
tar -xzvf etcd-v3.6.1-linux-amd64.tar.gz && mv etcd-v3.6.1-linux-amd64/etcdutl /usr/local/bin/etcdutl
```

You can check current `etcd` version using following command (might not work, if etcd and Kubernetes API are already unavailable):

```
d8 k -n kube-system exec -ti etcd-$(hostname) -- etcdutl version
```

2. Stop the etcd.

```
mv /etc/kubernetes/manifests/etcd.yaml ~/etcd.yaml
```

3. Save the current etcd data.

```
cp -r /var/lib/etcd/member/ /var/lib/deckhouse-etcd-backup
```

4. Copy or move the `etcd-backup.snapshot` file to the current user’s (root) home directory:

```
cp /path/to/backup/etcd-backup.snapshot ~/etcd-backup.snapshot
```

5. Clean the etcd directory.

```
rm -rf /var/lib/etcd
```

6. Restore the etcd database.

```
ETCDCTL_API=3 etcdutl snapshot restore ~/etcd-backup.snapshot  --data-dir=/var/lib/etcd
```

7. Run etcd. The process may take some time.

```
mv ~/etcd.yaml /etc/kubernetes/manifests/etcd.yaml
crictl ps --label io.kubernetes.pod.name=etcd-$HOSTNAME
```

To verify that etcd is running, use the command:

```
crictl ps --label io.kubernetes.pod.name=etcd-$HOSTNAME
```

Output example:

```
CONTAINER        IMAGE            CREATED              STATE     NAME      ATTEMPT
POD ID          POD
4b11d6ea0338f    16d0a07aa1e26    About a minute ago   Running   etcd      0
ee3c8c7d7bba6   etcd-gs-test
```

8. Restart the master node.

Restoring a multi-master cluster

To properly restore a multi-master cluster, follow these steps:

1. Enable High Availability (HA) mode. This is necessary to preserve at least one Prometheus replica and its PVC, since HA is disabled by default in single-master clusters.

2. Switch the cluster to single-master mode:
   - In a cloud cluster, follow the instructions.
   - In a static cluster, remove any unnecessary master nodes from the `control-plane` role by following the instructions and then remove them from the cluster.
   - In a static cluster with the configured HA mode based on two master nodes and an arbiter node, remove the arbiter node and additional master nodes.
   - In a cloud cluster with the configured HA mode based on two master nodes and an arbiter node, use the instructions to remove the additional master nodes and the arbiter node.
3. Restore etcd from the backup on the only remaining master node. Follow the instructions for restoring a cluster with a single master node.
4. Once etcd is restored, remove the records of the previously deleted master nodes from the cluster using the following command (replace with the actual node name):

   ```
   d8 k delete node <MASTER_NODE_NAME>
   ```

   > **Warning.** If the `d8 k` or `kubectl` commands are unavailable on the node, check the `/etc/kubernetes/kubernetes-api-proxy/nginx.conf` configuration file. It should only specify your current API server. If the configuration contains lines with IP addresses of old master nodes, remove them. Edit the configuration in a similar way on all other nodes.

5. Reboot the master node. Ensure that the other nodes transition to the `Ready` state.
6. Wait for Deckhouse to process all tasks in the queue:

   ```
   d8 system queue main
   ```

7. Switch the cluster back to multi-master mode. Follow the respective instructions for cloud clusters and static clusters.

Once you go through these steps, the cluster will be successfully restored in the multi-master configuration.

How do I restore a Kubernetes object from an etcd backup?

To get cluster objects data from an etcd backup, you need:

1. Start an temporary instance of etcd.
2. Fill it with data from the backup.
3. Get desired objects using `auger`.

Example of steps to restore objects from an etcd backup

In the example below, `etcd-backup.snapshot` is an etcd snapshot, `infra-production` is the namespace in which objects need to be restored.

- To decode objects from `etcd` you would need auger. It can be built from source on any machine that has Docker installed (it cannot be done on cluster nodes).

  ```
  git clone -b v1.0.1 --depth 1 https://github.com/etcd-io/auger
  cd auger
  make release
  build/auger -h
  ```

- Resulting executable `build/auger`, and also the `snapshot` from the backup copy of etcd must be uploaded on master-node, on which following actions would be performed.

Following actions are performed on a master node, to which `etcd snapshot` file and `auger` tool were copied:

1. Set the correct access permissions for the backup file:

   ```
   chmod 644 etcd-backup.snapshot
   ```

2. Set full path for snapshot file and for the tool into environmental variables:

   ```
   SNAPSHOT=/root/etcd-restore/etcd-backup.snapshot
   AUGER_BIN=/root/auger
   chmod +x $AUGER_BIN
   ```

3. Run a Pod with temporary instance of `etcd`.

- Create Pod manifest. It should schedule on current master node by `$HOSTNAME` variable, and mounts snapshot file by `$SNAPSHOT` variable, which it then restores in temporary `etcd` instance:

```
cat <<EOF >etcd.pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: etcdrestore
  namespace: default
spec:
  nodeName: $HOSTNAME
  tolerations:
  - operator: Exists
  initContainers:
  - command:
    - etcdutl
    - snapshot
    - restore
    - "/tmp/etcd-snapshot"
    - --data-dir=/default.etcd
    image: $(kubectl -n kube-system get pod -l component=etcd -o jsonpath="
{.items[*].spec.containers[*].image}" | cut -f 1 -d ' ')
    imagePullPolicy: IfNotPresent
    name: etcd-snapshot-restore
    # Uncomment the fragment below to set limits for the container if the node does
not have enough resources to run it.
    # resources:
    #   requests:
    #     ephemeral-storage: "200Mi"
    #   limits:
    #     ephemeral-storage: "500Mi"
    volumeMounts:
    - name: etcddir
      mountPath: /default.etcd
    - name: etcd-snapshot
      mountPath: /tmp/etcd-snapshot
      readOnly: true
  containers:
  - command:
    - etcd
    image: $(kubectl -n kube-system get pod -l component=etcd -o jsonpath="
{.items[*].spec.containers[*].image}" | cut -f 1 -d ' ')
    imagePullPolicy: IfNotPresent
    name: etcd-temp
    volumeMounts:
    - name: etcddir
      mountPath: /default.etcd
  volumes:
  - name: etcddir
    emptyDir: {}
    # Use the snippet below instead of emptyDir: {} to set limits for the container
if the node's resources are insufficient to run it.
    # emptyDir:
    #  sizeLimit: 500Mi
  - name: etcd-snapshot
    hostPath:
      path: $SNAPSHOT
      type: File
EOF
```

- Create Pod from the resulting manifest:

```
d8 k create -f etcd.pod.yaml
```

4. Set environment variables. In this example:
   - `infra-production` - namespace which we will search resources in.

- `/root/etcd-restore/output` - path for outputting recovered resource manifests.
- `/root/auger` - path to `auger` executable.

```
FILTER=infra-production
BACKUP_OUTPUT_DIR=/root/etcd-restore/output
mkdir -p $BACKUP_OUTPUT_DIR && cd $BACKUP_OUTPUT_DIR
```

5. Commands below will filter needed resources by `$FILTER` and output them into `$BACKUP_OUTPUT_DIR` directory:

```
files=($(d8 k -n default exec etcdrestore -c etcd-temp -- etcdctl  --endpoints=localhost:2379 get / --prefix --keys-only | grep "$FILTER"))
for file in "${files[@]}"
do
  OBJECT=$(d8 k -n default exec etcdrestore -c etcd-temp -- etcdctl  --endpoints=localhost:2379 get "$file" --print-value-only | $AUGER_BIN decode)
  FILENAME=$(echo $file | sed -e "s#/registry/##g;s#/#_#g")
  echo "$OBJECT" > "$BACKUP_OUTPUT_DIR/$FILENAME.yaml"
  echo $BACKUP_OUTPUT_DIR/$FILENAME.yaml
done
```

6. From resulting `yaml` files, delete `creationTimestamp`, `UID`, `status` and other operational fields, and then restore the objects:

```
d8 k create -f deployments_infra-production_supercronic.yaml
```

7. Delete the Pod with a temporary instance of etcd:

```
d8 k -n default delete pod etcdrestore
```

How the node to run the Pod on is selected

The Kubernetes scheduler component selects the node to run the Pod on.

The selection process involves two phases, namely `Filtering` and `Scoring`. They are supposed to efficiently distribute the Pods between the nodes.

Although there are some additional phases, such as `pre-filtering`, `post-filtering`, and so on, you can safely narrow them down to the global phases mentioned above, as they merely increase flexibility and help to optimize things.

The structure of the Kubernetes scheduler

The Scheduler comprises plugins that function in either or both phases.

Example of plugins:

- **ImageLocality** — favors nodes that already have the container images that the Pod runs. Phase: `Scoring`.
- **TaintToleration** — implements taints and tolerations. Phases: `Filtering`, `Scoring`.
- **NodePorts** - checks whether the ports required for the Pod to run are available on the node. Phase: `Filtering`.

The full list of plugins is available in the Kubernetes documentation.

Working logic

Scheduler profiles

There are two predefined scheduler profiles:

- `default-scheduler`: The default profile that distributes pods to nodes with the lowest load;
- `high-node-utilization`: A profile that places pods on nodes with the highest load.

To specify a scheduler profile, use the `spec.schedulerName` parameter in the pod manifest.

Example of using a profile:

```
apiVersion: v1
kind: Pod
metadata:
  name: scheduler-example
  labels:
    name: scheduler-example
spec:
  schedulerName: high-node-utilization
  containers:
  - name: example-pod
    image: registry.k8s.io/pause:2.0
```

Pod scheduling stages

The selection process starts with the `Filtering` phase. During it, `filter` plugins select nodes that satisfy filter conditions such as `taints`, `nodePorts`, `nodeName`, `unschedulable`, etc. If the nodes are in different zones, the scheduler alternates zones when selecting to ensure that all Pods will not end up in the same zone.

Suppose there are two zones with the following nodes:

```
Zone 1: Node 1, Node 2, Node 3, Node 4
Zone 2: Node 5, Node 6
```

In this case, the nodes will be selected in the following order:

```
Node 1, Node 5, Node 2, Node 6, Node 3, Node 4.
```

Note that Kubernetes limits the number of nodes to calculate their scores during scheduling. This optimizes the selection process and prevents unnecessary scoring. By default, the threshold is linear. For clusters with less than or equal to 50 nodes, 100% of nodes are considered for scheduling; for clusters with 100 nodes, a 50%-threshold is used; and for clusters with 5000 nodes, a 10%-threshold is used. The minimum threshold value is 5% for clusters with more than 5000 nodes. Therefore, even if all the conditions are met, a node may not be included in the list of candidates for scheduling if the default settings are used.

This logic can be changed (read more about the parameter `percentage Of Nodes To Score` in the Kubernetes documentation), but Deckhouse does not provide such an option.

The `Scoring` phase follows once the nodes that meet the conditions are selected. Each plugin evaluates the filtered node list and assigns a score to each node based on available resources: `pod capacity`, `affinity`, `volume provisioning`, and other factors. The scores from the different plugins are then summed up and the node with the highest score is selected. If several nodes have the same score, the node is selected at random.

Finally, the scheduler assigns the Pod to the node with the highest ranking.

Documentation

- General description of the scheduler.
- Plugin system.
- Node Filtering Details.
- Scheduler source code.

How to change or extend the scheduler logic

To change the logic of the scheduler it is possible to use the extension mechanism Extenders.

Each plugin is a webhook that must satisfy the following requirements:

- Use of TLS.
- Accessibility through a service within the cluster.
- Support for standard `Verbs` (`filterVerb = filter`, `prioritizeVerb = prioritize`).
- It is also assumed that all plugins can cache node information (`nodeCacheCapable: true`).

You can connect an `extender` using KubeSchedulerWebhookConfiguration resource.

✖ When using the `failurePolicy: Fail` option, in case of an error in the webhook's operation, the scheduler will stop working and new pods will not be able to start.

How does kubelet certificate rotation work?

In Deckhouse Kubernetes Platform, kubelet certificate rotation is automatic.

The kubelet uses a client TLS certificate (`/var/lib/kubelet/pki/kubelet-client-current.pem`) with which it can request a new client certificate or a new server certificate (`/var/lib/kubelet/pki/kubelet-server-current.pem`) from kube-apiserver.

When there is 5-10% (random value from the range) of time left before the certificate expires, kubelet requests a new certificate from kube-apiserver. For a description of the algorithm, see the official Kubernetes documentation.

Certificates lifetime

By default, lifetime of certificates is 1 year (8760 hours). If necessary, this value can be changed using `--cluster-signing-duration` argument in `/etc/kubernetes/manifests/kube-controller-manager.yaml` manifest. But to ensure that kubelet has time to install the certificate before it expires, we recommend setting the certificate lifetime to more than 1 hour.

> ⚠ If the client certificate lifetime has expired, kubelet will not be able to make requests to kube-apiserver and will not be able to renew certificates. In this case, the node will be marked as `NotReady` and recreated.

Specifics of working with kubelet server certificates in Deckhouse Kubernetes Platform

Deckhouse Kubernetes Platform uses IP addresses for kubelet API requests. The kubelet configuration does not use the `tlsCertFile` and `tlsPrivateKeyFile` fields, but uses a dynamic certificate that kubelet generates itself. Also, the CIS benchmark `AVD-KCV-0088` and `AVD-KCV-0089` checks, which track whether the `--tls-cert-file` and `--tls-private-key-file` arguments were passed to kubelet, are disabled in the Deckhouse Kubernetes Platform (in the `operator-trivy` module).

Information about the logic of working with server certificates in Kubernetes

Kubelet handles server certificates using the following logic:

- If `tlsCertFile` and `tlsPrivateKeyFile` are not empty, kubelet will use them as the default certificate and key.
  - When a client requests the kubelet API by specifying an IP address (e.g., `https://10.1.1.2:10250/`), the default private key (`tlsPrivateKeyFile`) will be used to establish a TLS connection. In this case, certificate rotation will not work.
  - When a client requests the kubelet API by specifying a host name (e.g., `https://k8s-node:10250/`), a dynamically generated private key from the `/var/lib/kubelet/pki/` directory will be used to establish a TLS connection. In this case, certificate rotation will work.
- If `tlsCertFile` and `tlsPrivateKeyFile` are empty, a dynamically generated private key from the `/var/lib/kubelet/pki/` directory will be used to establish the TLS connection. In this case, certificate rotation will work.

How to manually update control plane component certificates?

There may be a situation when the cluster’s master nodes are powered off for an extended period. During this time, the control plane component certificates may expire. After the nodes are powered back on, the certificates will not update automatically and must be renewed manually.

Control plane component certificates are updated using the `kubeadm` utility. To update the certificates, do the following on each master node:

1. Find the `kubeadm` utility on the master node and create a symbolic link using the following command:

   ```
   ln -s $(find /var/lib/containerd -name kubeadm -type f -executable -print -quit) /usr/bin/kubeadm
   ```

2. Update the certificates:

   ```
   kubeadm certs renew all
   ```

How do I protect sensitive fields in custom resources?

To protect sensitive fields (such as passwords, tokens, or keys) in resource schemas from unauthorized access via the API, unencrypted storage in etcd, or exposure in audit logs, use the `CRDSensitiveData` feature gate together with the `x-kubernetes-sensitive-data` schema marker.

To enable field protection, do the following:

1. Enable etcd encryption using the `apiserver.encryptionEnabled` parameter in the module settings. The `CRDSensitiveData` feature gate is enabled automatically along with the encryption and shouldn’t be specified manually.

> ⚠ Enabling `apiserver.encryptionEnabled` is irreversible and triggers a `kube-apiserver` restart.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  version: 2
  enabled: true
  settings:
    apiserver:
      encryptionEnabled: true
```

2. Mark sensitive fields in the resource schema with `x-kubernetes-sensitive-data: true`.

   The marker can be applied to the fields of the following types: `string`, `integer`, `number`, `boolean`, `object` and `array` (applying the marker to an `object` or `array` makes the entire subtree sensitive). Fields marked with `x-kubernetes-int-or-string: true` are supported as well.

   The marker can't be set on the schema root (the `openAPIV3Schema` node) and inside `anyOf`, `oneOf`, `allOf`, or `not` branches.

3. For users and ServiceAccounts that require access to full data, grant RBAC permissions for the `<resource>/sensitive` subresource.

Protection mechanisms

Protection	Description
Encryption in etcd	The entire resource is encrypted using the same AES-CBC transformer as Kubernetes Secrets.
API field filtering	Sensitive fields are removed from responses to `get`, `list`, and `watch` requests if the caller does not have permissions on the `<resource>/sensitive` subresource.
Audit log masking	Values of sensitive fields are replaced with `"******"` in all audit events, regardless of RBAC permissions and audit level.

A complete configuration example and results are available in the Examples section.

Examples

> **i** The module lifecycle stage: General Availability

The connection of an external scheduler plugin

Example of connecting an external scheduler plugin via webhook.

```
apiVersion: deckhouse.io/v1alpha1
kind: KubeSchedulerWebhookConfiguration
metadata:
  name: sds-replicated-volume
webhooks:
- weight: 5
  failurePolicy: Ignore
  clientConfig:
    service:
      name: scheduler
      namespace: d8-sds-replicated-volume
      port: 8080
      path: /scheduler
    caBundle: ABCD=
  timeoutSeconds: 5
```

Protecting resources with sensitive fields

The following example demonstrates how to protect sensitive fields in resources using the `CRDSensitiveData` feature gate and the `x-kubernetes-sensitive-data` schema marker.

For instructions on enabling this feature, see FAQ.

1. Enabling etcd encryption with the `encryptionEnabled` parameter. This automatically enables the `CRDSensitiveData` feature gate for `kube-apiserver`.

   ```
   apiVersion: deckhouse.io/v1alpha1
   kind: ModuleConfig
   metadata:
     name: control-plane-manager
   spec:
     version: 2
     enabled: true
     settings:
       apiserver:
         encryptionEnabled: true
   ```

2. Defining sensitive fields in the resource schema.

   Fields marked with `x-kubernetes-sensitive-data: true` are encrypted in etcd and removed from API responses for callers that do not have access to the `<resource>/sensitive` subresource.

   ```
   apiVersion: apiextensions.k8s.io/v1
   kind: CustomResourceDefinition
   metadata:
     name: dbconfigs.example.com
   spec:
     group: example.com
     scope: Namespaced
     names:
       plural: dbconfigs
       singular: dbconfig
       kind: DbConfig
     versions:
       - name: v1
         served: true
         storage: true
         schema:
           openAPIV3Schema:
             type: object
             properties:
               spec:
                 type: object
                 properties:
                   host:
                     type: string
                   username:
                     type: string
                   password:
                     type: string
                     x-kubernetes-sensitive-data: true
   ```

3. Creating a custom resource with values filled in sensitive fields.

   ```
   apiVersion: example.com/v1
   kind: DbConfig
   metadata:
     name: primary
     namespace: default
   spec:
     host: db.example.com
     username: admin
     password: s3cr3t
   ```

   Once saved, the entire object is encrypted in etcd and the `password` value is hidden in the audit log and

removed from the API responses of the caller has no permissions to the `dbconfigs/sensitive` subresource.

4. Configuring access to sensitive fields using RBAC and the `<resource>/sensitive` subresource.

```
# Regular role: can read the resource, but sensitive fields are removed from responses.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: dbconfig-reader
rules:
- apiGroups: ["example.com"]
  resources: ["dbconfigs"]
  verbs: ["get", "list", "watch"]
---
# Privileged role: can read full data, including sensitive fields.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: dbconfig-sensitive-reader
rules:
- apiGroups: ["example.com"]
  resources: ["dbconfigs"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["example.com"]
  resources: ["dbconfigs/sensitive"]
  verbs: ["get", "list", "watch"]
```

5. A result of sensitive field protection.
   - A user with the `dbconfig-reader` role who runs `d8 k get dbconfig primary -o json` can see the resource with sensitive fields removed:

     ```
     {
       "spec": {
         "host": "db.example.com",
         "username": "admin"
       }
     }
     ```

   - A user with the `dbconfig-sensitive-reader` role can see the full data:

     ```
     {
       "spec": {
         "host": "db.example.com",
         "username": "admin",
         "password": "s3cr3t"
       }
     }
     ```

   - In audit logs, values of sensitive fields are always masked, regardless of caller permissions:

     ```
     {
       "spec": {
         "host": "db.example.com",
         "username": "admin",
         "password": "******"
       }
     }
     ```

The csi-vsphere module

✖ The module lifecycle stage: Deprecated

The `csi-vsphere` module provides Container Storage Interface (CSI) support for VMware vSphere environments, enabling dynamic provisioning and management of persistent storage volumes in Kubernetes clusters running

on vSphere infrastructure.

This module is specifically designed for **static Kubernetes clusters** (non-cloud) deployed on VMware vSphere and works independently from the cloud-provider-vsphere module. It integrates with vSphere datastores to provide persistent storage capabilities through the vSphere CSI driver.

> **i** Use the `csi-vsphere` module when the cluster cannot use the `cloud-provider-vsphere` module but still requires dynamic volume provisioning on vSphere infrastructure.

Main Features

- Provides dynamic persistent volume provisioning for static Kubernetes clusters on VMware vSphere.
- Automatically discovers vSphere datastores and creates StorageClass resources.
- Supports topology-aware volume placement using vSphere region and zone tags.
- Handles volume creation, deletion, attachment, detachment, and online resizing (Default mode).
- Supports Legacy, Default, and Migration compatibility modes for smooth driver transitions.

The module deploys the VMware vSphere CSI driver components that:

- **Automatically discover vSphere datastores**: Reads discovery data from the `d8-cloud-provider-discovery-data` secret to identify available datastores and their topology.
- **Create StorageClasses**: Automatically generates StorageClass resources for each discovered datastore, filtered by the exclude list if configured.
- **Provision persistent volumes**: Dynamically creates persistent volumes on vSphere datastores when PersistentVolumeClaims are created.
- **Support volume operations**: Handles volume creation, deletion, attachment, detachment, and online resizing (in Default mode).
- **Implement topology awareness**: Uses zone and region labels for proper volume placement according to vSphere cluster topology.

System requirements

To use the module in a cluster, the following system requirements must be met:

- All cluster nodes must be virtual machines created with vSphere tooling.
- The vSphere VM name must **exactly match** the node hostname in the Deckhouse Kubernetes Platform cluster.
- Each VM must have `disk.EnableUUID:TRUE` set so the platform can identify attached volumes reliably.

Architecture

The CSI driver consists of components:

1. **CSI Controller** (runs on master nodes):
   - **vsphere-csi-controller**: Main controller managing volume lifecycle operations;
   - **csi-provisioner**: Watches for PVCs and triggers volume creation;
   - **csi-attacher**: Handles volume attachment to nodes;
   - **csi-resizer**: Manages online volume expansion;
   - **vsphere-syncer**: Synchronizes metadata between Kubernetes and vSphere every 30 minutes;
   - **liveness-probe**: Monitors controller health.
2. **CSI Node** (DaemonSet on all nodes):
   - **vsphere-csi-node**: Node-level CSI driver handling volume mounting/unmounting;
   - **node-driver-registrar**: Registers the CSI driver with kubelet;
   - **liveness-probe**: Monitors node driver health.

Compatibility modes

The module supports the following operational modes (the mode is set via the `compatibilityFlag` parameter):

- **Default mode** (used by default, `compatibilityFlag` not set or empty):
  - uses the current vSphere CSI driver (`csi.vsphere.vmware.com`);
  - supports CNS (Cloud Native Storage) volumes;
  - enables online volume resizing;
  - recommended for new deployments.
- **Legacy mode** (`compatibilityFlag: "Legacy"`):
  - uses the older vSphere CSI driver (`vsphere.csi.vmware.com`);
  - works with FCD (First Class Disk) volumes only;
  - no online volume resizing support;
  - used for backward compatibility.
- **Migration mode** (`compatibilityFlag: "Migration"`):
  - runs both driver versions simultaneously,
  - facilitates migration from Legacy to Default mode,
  - allows gradual transition of workloads.

Storage discovery and provisioning

The module automatically:

1. **Discovers datastores** from the cloud provider discovery data.
2. **Generates StorageClass names** by sanitizing datastore names to meet Kubernetes DNS naming requirements.
3. **Filters excluded storage** based on storageClass.exclude configuration (supports regex patterns).
4. **Creates StorageClasses** with appropriate parameters:
   - `allowVolumeExpansion: true` (Default mode only).
   - `volumeBindingMode: WaitForFirstConsumer`: Ensures volumes are created in the same zone as the pod.
   - Topology constraints matching vSphere regions and zones.

Default StorageClass and exclusions

The module automatically creates a StorageClass for each Datastore and DatastoreCluster from the zones.

To set the default StorageClass for a cluster, use the global parameter `defaultClusterStorageClass`. To filter out unwanted StorageClass objects, specify exclusion patterns in the `storageClass.exclude` parameter.

Volume lifecycle

When a pod requests storage:

1. A PVC is created referencing a StorageClass managed by this module.
2. The CSI provisioner creates a volume on the specified vSphere datastore.
3. The CSI attacher attaches the volume to the appropriate node.
4. The CSI node driver mounts the volume into the pod.
5. The vsphere-syncer periodically synchronizes metadata to ensure consistency.

Limitations

- This module is designed exclusively for static (non-cloud) Kubernetes clusters on vSphere.
- Legacy mode does not support online volume resizing.
- Volume expansion requires the current CSI driver (`csi.vsphere.vmware.com`).
- StorageClasses are automatically managed — manual modifications may be overwritten.

Configuration

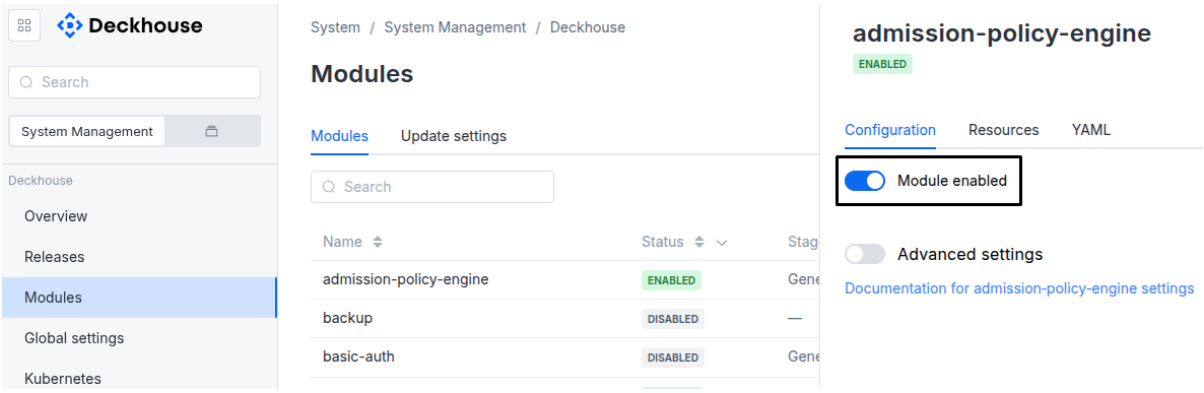
> ✖ The module lifecycle stage: Deprecated

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `csi-vsphere` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable csi-vsphere
```

- **Using ModuleConfig `csi-vsphere`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `csi-vsphere` (create it if necessary);

Example of a manifest to enable module `csi-vsphere`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-vsphere
spec:
  enabled: true
```

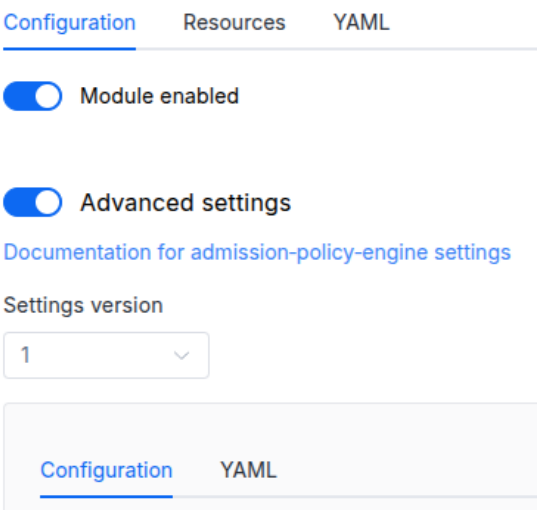
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `csi-vsphere` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `csi-vsphere` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `csi-vsphere`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `csi-vsphere` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc csi-vsphere
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `csi-vsphere` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `csi-vsphere`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: csi-vsphere
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.disableTimesync**
    boolean

    Disable time synchronization on the vSphere side.

    > **Warning.** This parameter does not disable NTP daemons in the guest OS; it only disables time correction on the ESXi side.

  - **settings.host**
    string

    Required value

    The domain of the vCenter server.

    Example:

    ```
    host: myhost
    ```

  - **settings.insecure**
    boolean

    Set to `true` if vCenter has a self-signed certificate.

  - **settings.password**
    string

    Required value

    The user’s password.

    Example:

    ```
    password: myPaSsWd
    ```

  - **settings.region**
    string

    Required value

    Is a tag added to the vSphere Datacenter where all actions will occur: provisioning VirtualMachines, storing virtual disks on datastores, connecting to the network.

    Example:

    ```
    region: myreg
    ```

  - **settings.regionTagCategory**
    string

    Required value

    The name of the tag category used to identify the region (vSphere Datacenter).

    Example:

    ```
    regionTagCategory: myregtagcat
    ```

  - **settings.storageClass**
    object

- **settings.storageClass.compatibilityFlag**
  string

  A flag allowing the use of the old CSI version:

  - `Legacy` — use the old version of the driver. FCD discs only, no online-resizing;
  - `Migration` — in this case, both drivers will be available in the cluster at the same time. This mode is used to migrate from an old driver.

  Allowed values: `Legacy`, `Migration`

- **settings.storageClass.exclude**
  array of strings

  A list of StorageClass names (or regex expressions for names) to exclude from the creation in the cluster.

  Example:

  ```
  exclude:
  - ".*-lun101-.*"
  - slow-lun103-1c280603
  ```

- **settings.username**
  string

  Required value

  The login.

  Example:

  ```
  username: myuname
  ```

- **settings.vmFolderPath**
  string

  Required value

  The path to the VirtualMachine Folder where the cloned VMs will be created.

  Example:

  ```
  vmFolderPath: dev/test
  ```

- **settings.zoneTagCategory**
  string

  Required value

  The name of the tag category used to identify the zone (vSphere Cluster).

  Example:

  ```
  zoneTagCategory: myzonetagcat
  ```

- **settings.zones**
  array of strings

  Required value

  The globally restricted set of zones that this Cloud Provider works with.

  Example:

  ```
  zones:
  - zonea
  - zoneb
  ```

Preparing environment

✖ The module lifecycle stage: Deprecated

This section describes how to prepare vCenter and vSphere for the `csi-vsphere` module.

Required resources

The following resources are required for the module to function:

- User with the required permissions.
- Network with DHCP and internet access.
- Datacenter tagged as `k8s-region`.
- Cluster tagged as `k8s-zone`.
- One or more Datastores with the appropriate tags.

Installing govc

Use the vSphere CLI govc.

Set environment variables for vCenter access:

```
export GOVC_URL=example.com
export GOVC_USERNAME=<username>@vsphere.local
export GOVC_PASSWORD=<password>
export GOVC_INSECURE=1
```

Create tags and tag categories

`csi-vsphere` maps Kubernetes topology to vSphere objects: a **region** is a vSphere Datacenter, and a **zone** is a vSphere Cluster. The relationship between these objects is defined using tags.

To link Cluster and Datacenter objects, follow these steps:

1. Create tag categories:

   ```
   govc tags.category.create -d "Kubernetes Region" k8s-region
   govc tags.category.create -d "Kubernetes Zone" k8s-zone
   ```

2. Create tags in each category. If you use several zones (Cluster), create one tag per cluster:

   ```
   govc tags.create -d "Kubernetes Region" -c k8s-region test-region
   govc tags.create -d "Kubernetes Zone Test 1" -c k8s-zone test-zone-1
   govc tags.create -d "Kubernetes Zone Test 2" -c k8s-zone test-zone-2
   ```

3. Attach the region tag to the Datacenter:

   ```
   govc tags.attach -c k8s-region test-region /<DatacenterName>
   ```

4. Attach zone tags to Cluster objects:

   ```
   govc tags.attach -c k8s-zone test-zone-1 /<DatacenterName>/host/<ClusterName1>
   govc tags.attach -c k8s-zone test-zone-2 /<DatacenterName>/host/<ClusterName2>
   ```

Datastore configuration

> **i** For dynamic PersistentVolume provisioning, each Datastore must be reachable from every ESXi host (shared datastore).

Attach region and zone tags to Datastore objects so the module can create StorageClass resources automatically:

```
govc tags.attach -c k8s-region test-region /<DatacenterName>/datastore/<DatastoreName1>
govc tags.attach -c k8s-zone test-zone-1 /<DatacenterName>/datastore/<DatastoreName1>

govc tags.attach -c k8s-region test-region /<DatacenterName>/datastore/<DatastoreName2>
govc tags.attach -c k8s-zone test-zone-2 /<DatacenterName>/datastore/<DatastoreName2>
```

Create and assign the role

> **i** Due to the variety of SSO providers connected to csi-vsphere, the steps for creating a user are not covered in this article.
>
> The role to be created below includes all possible privileges for all DKP components. For a detailed list of privileges, refer to the `cloud-provider-vsphere` documentation.

Create the role:

```
govc role.create deckhouse \
   Cns.Searchable Datastore.AllocateSpace Datastore.Browse Datastore.FileManagement \
   Global.GlobalTag Global.SystemTag Network.Assign StorageProfile.View \
   $(govc role.ls Admin | grep -F -e 'Folder.' -e 'InventoryService.' -e 'Resource.' -e
'VirtualMachine.')
```

Grant the role on the `vCenter` root object:

```
govc permissions.set -principal <username>@vsphere.local -role deckhouse /
```

The deckhouse module

> **i** The module lifecycle stage: General Availability

In Deckhouse, this module sets up:

- **The logging level**;
- **The set of modules enabled by default**;

  Usually, the `Default` set is used (it is suitable for most cases).

  Regardless of the set of modules enabled by default, any module can be explicitly enabled or disabled in the Deckhouse configuration (learn more about enabling and disabling a module).
- **The release channel**;

  Deckhouse has a built-in mechanism for automatic updates. This mechanism uses 5 release channels with various stability and frequency of releases. Learn more about how the automatic update mechanism works and how you can set the desired release channel
- **The update mode** and **update windows**;

  Deckhouse supports **manual** and **automatic** update modes.

  In the manual upgrade mode, only critical fixes (patch releases) are automatically applied, and upgrading to a more current Deckhouse release requires manual confirmation.

  In the automatic update mode, Deckhouse switches to a newer release as soon as it is available in the corresponding release channel unless update windows are **configured** for the cluster. If update windows are **configured** for the cluster, Deckhouse will upgrade to a newer release during the next available update window.
- **Service for validating Custom Resources**.

  The validation service prevents creating Custom Resources with invalid values or adding such values to the existing Custom Resources. Note that it only tracks Custom Resources managed by Deckhouse modules.

Deckhouse releases update

Get Deckhouse releases status

You can get Deckhouse releases list via the command `d8 k get deckhousereleases`. By default, a cluster keeps the last 10 Superseded releases and all deployed/pending releases.

Every release can have one of the following statuses:

- `Pending` - release is waiting to be deployed: waiting for update window, canary deployment, etc. You can see the detailed status via the `d8 k describe deckhouserelease $name` command.
- `Deployed` - release is applied. It means that the image tag of the Deckhouse Pod was changed, but the update process of all components is going asynchronously and could not have been finished yet.
- `Superseded` - release is outdated and not used anymore.
- `Suspended` - release was suspended (for example, due to an error found). A release can have this status if it was suspended before being deployed in the cluster.

Update process

When release status is changed to `Deployed` state, release is updating only a tag of the Deckhouse image. Deckhouse will start checking and updating process of the all modules, which were changed from the last release. Duration of an update could be different and connected to cluster size, enabled modules count and settings. Foe example: if you cluster have a lot of `NodeGroup` resources, it will take some time to update them because these resources are updated one by one `IngressNginxControllers` also updating one by one.

Manual release deployment

If you have a manual update mode enabled and have a few Pending releases, you can approve them all at once. In that case Deckhouse will update in series keeping a release order and changing their status during the update.

Pin a release

To pin a release means fully or partially disable the automatic Deckhouse version update.

There are three options to limit the automatic update of Deckhouse:

- Enable a manual update mode.

  In this case, Deckhouse holds on a current version and able to receive updates. But to apply the update, a manual action will need to be performed. This applies to both patch versions and minor versions.

  To enable manual update mode, you need to set the parameter settings.update.mode in ModuleConfig `deckhouse` to `Manual`:

  ```
  d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":{"mode":"Manual"}}}}'
  ```

- Set the automatic update mode for patch versions.

  In this case, Deckhouse holds on a current version and will automatically update to patch versions of the current release (taking into account the update windows). To apply a minor version update, a manual action will need to be performed.

  For example: the current version of DKP is `v1.70.1`, after setting the automatic update mode for patch versions, Deckhouse can be updated to version `v1.70.2`, but will not update to version `v1.71.*` or higher.

  To set the automatic update mode for patch versions, you need to set the parameter settings.update.mode to `AutoPatch` in the ModuleConfig `deckhouse`:

  ```
  d8 k patch mc deckhouse --type=merge -p='{"spec":{"settings":{"update":
  {"mode":"AutoPatch"}}}}'
  ```

- Set a specified image tag for Deployment `deckhouse` and remove releaseChannel parameter from `deckhouse` module configuration.

  In that case, DKP holds on a current version, and no information about new available versions in the cluster (DeckhouseRelease objects) will be received.

  An example of installing version `v1.66.3` for DKP EE and removing the `releaseChannel` parameter from the configuration of the `deckhouse` module:

  ```
  d8 k -ti -n d8-system exec svc/deckhouse-leader -c deckhouse -- kubectl set image deployment/deckhouse
  deckhouse=registry.deckhouse.io/deckhouse/ee:v1.66.3
  d8 k patch mc deckhouse --type=json -p='[{"op": "remove", "path":
  "/spec/settings/releaseChannel"}]'
  ```

Priority Classes

This module creates a set of priority classes and assigns them to components installed by Deckhouse and applications in the cluster.

Priority classes are utilized by the scheduler to determine a Pod’s priority based on the class it belongs to during scheduling.

For example, when deploying Pods with `priorityClassName: production-low`, if the cluster lacks sufficient resources, Kubernetes will start evicting Pods with the lowest priority to accommodate the `production-low` Pods. That is, Kubernetes will first evict all Pods with `priorityClassName: develop` Pods, then proceed to `cluster-low` Pods, and so on.

When setting the priority class, it is crucial to understand the type of application and the environment it operates in. Assigning any priority class does not lower a Pod’s priority because Pods without a specified priority class are considered to have the lowest priority.

⚠ You cannot use the following priority classes: `system-node-critical`, `system-cluster-critical`, `cluster-medium`, `cluster-low`.

Below is the list of priority classes set by the module (sorted by the priority from highest to lowest):

Priority class	Description	Value
`system-node-critical`	Cluster components that must be present on the node. This priority class fully protects components against eviction by kubelet. Examples: `node-exporter`, `csi`, and others.	2000001000
`system-cluster-critical`	Cluster components that are critical to its correct operation. This PriorityClass is mandatory for MutatingWebhooks and Extension API servers. It also fully protects components against eviction by kubelet. Examples: `kube-dns`, `kube-proxy`, `cni-flannel`, `cni-cilium`, and others.	2000000000
`production-high`	Stateful applications in the production environment. Their unavailability leads to service downtime or data loss. Examples: `PostgreSQL`, `Memcached`, `Redis`, `MongoDB`, and others.	9000
`cluster-medium`	Cluster components responsible for monitoring (alerts, diagnostic tools) and autoscaling. Without monitoring, assessing the scale of incidents is impossible; without autoscaling, applications cannot receive necessary resources. Examples: `deckhouse`, `node-local-dns`, `grafana`, `upmeter`, and others.	7000
`production-medium`	Main stateless applications in the production environment that are responsible for operating the service for end-users.	6000
`deployment-machinery`	Cluster components that are responsible for deploying/building.	5000
`production-low`	Non-critical, secondary applications in the production environment (crons, admin dashboards, batch processing). For important batch or cron jobs, consider assigning them the `production-medium` priority.	4000
`staging`	Staging environments for applications.	3000
`cluster-low`	Cluster components that are desirable but not essential for proper cluster operation. Examples: `dashboard`, `cert-manager`, `prometheus`, and others.	2000
`develop` (default)	Develop environments for applications. The default class for a component (if other priority classes aren’t set).	1000
`standby`	This class is not intended for applications. It is used for system purposes (reserving nodes).	-1

Third-party components

List of third-party software used in the `deckhouse` module:

- Shell operator 1.9.3

  License: Apache License 2.0

  A tool for running event-driven scripts in a Kubernetes cluster

- JQ 1.7.1

  License: Mixed (https://github.com/stedolan/jq/blob/master/COPYING)

  A lightweight and flexible command-line JSON processor.

Configuration

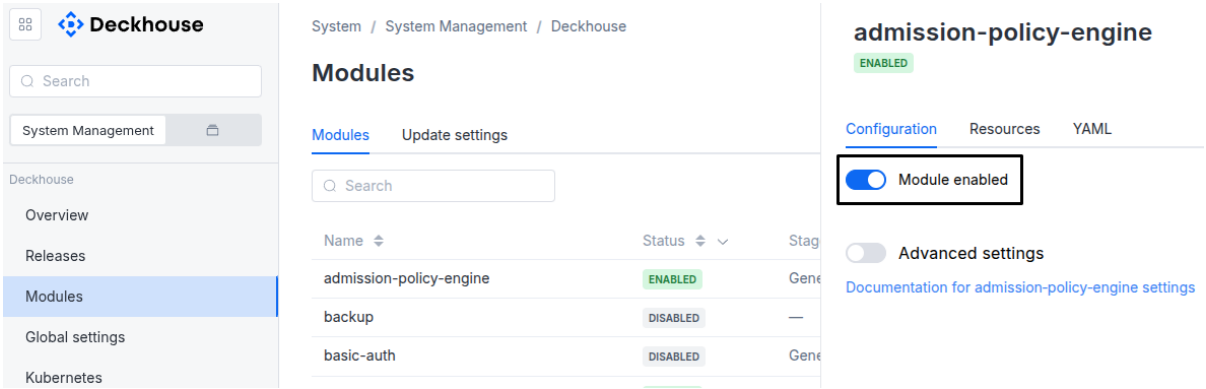
> The module lifecycle stage: General Availability

The module is **enabled** by default in the following bundles: `Default`, `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `deckhouse` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable deckhouse
```

- **Using ModuleConfig `deckhouse`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `deckhouse` (create it if necessary);

  Example of a manifest to enable module `deckhouse`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  enabled: true
```

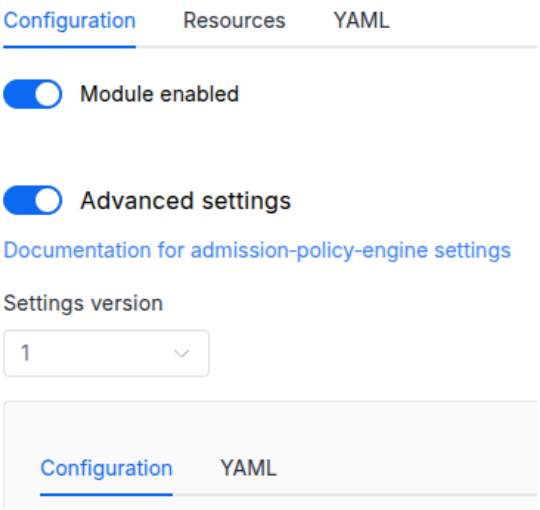
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `deckhouse` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `deckhouse` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `deckhouse`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `deckhouse` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc deckhouse
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `deckhouse` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `deckhouse`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: deckhouse
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.allowExperimentalModules**
    boolean

    Allows the use of experimental modules (modules in the `Experimental` lifecycle stage).

    Experimental modules can be unstable and are not supported under the Deckhouse SLA. Use them with caution.

    Default: `false`

    Example:

    ```
    allowExperimentalModules: false
    ```

  - **settings.bundle**

string

The set of DKP modules that are enabled by default.

Changing the parameter value in an already deployed cluster is not possible.

Possible values:

- `Default`: Recommended set of modules for cluster operation. Includes modules for monitoring, authorization control, networking, and other basic platform needs.
- `Managed`: Set of modules for managed clusters from cloud providers.
- `Minimal`: Minimum possible set, which includes a single module (this one).

  Warning. Several basic platform modules (for example, the CNI module) are not included in the `Minimal` module set. Without the basic modules, DKP can only operate in an already deployed cluster. Read the “Things to keep in mind when working with the Minimal module set” section for more details.

Default: `Default`

Allowed values: `Default`, `Minimal`, `Managed`

Example:

```
bundle: Default
```

- **settings.highAvailability**
  boolean

  Manually enable the high availability mode.

  By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

  Example:

  ```
  highAvailability: true
  ```

- **settings.license**
  object

  Edition settings of the Deckhouse Kubernetes Platform.

  - **settings.license.edition**
    string

    Edition of the Deckhouse Kubernetes Platform:

    - `CE` — Community Edition.
    - `BE` - Basic edition.
    - `EE` — Enterprise Edition.
    - `SE` — Standard Edition.
    - `SE-plus` — Standard Edition+.
    - `CSE` — Certified Security Edition.

    Default: `CE`

    Example:

    ```
    edition: CE
    ```

- **settings.logLevel**
  string

  Deckhouse logging level.

  Default: `Info`

  Allowed values: `Trace`, `Debug`, `Info`, `Error`

  Example:

  ```
  logLevel: Info
  ```

- **settings.nodeSelector**
  object

  The same as in the Pods’ `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, `nodeSelector` will be determined automatically.

Caution! Deckhouse will stop working if there is a nonexistent label in `nodeSelector`. You need to change the values to the correct ones in `ModuleConfig/deckhouse` and `deployment/deckhouse` to get Deckhouse back on track.

- **settings.registry**
  object

  Settings for accessing the container registry with Deckhouse images.

  This parameter allows configuring how DKP connects to the container registry to pull its own component images. Settings are processed by the internal registry module. In clusters where this configuration is not supported, the registry connection setup is done manually using the `change-registry` helper script.

  Examples:

  ```
  registry:
    mode: Unmanaged
  ```

  ```
  registry:
    mode: Unmanaged
    unmanaged:
      imagesRepo: registry.deckhouse.io/deckhouse/ee
      license: DECKHOUSE_LICENSE_KEY
  ```

  ```
  registry:
    mode: Direct
    direct:
      imagesRepo: registry.deckhouse.io/deckhouse/ee
      license: DECKHOUSE_LICENSE_KEY
  ```

  ```
  registry:
    mode: Proxy
    proxy:
      imagesRepo: registry.deckhouse.io/deckhouse/ee
      license: DECKHOUSE_LICENSE_KEY
  ```

  ```
  registry:
    mode: Local
  ```

  - **settings.registry.direct**
    object

    Parameters for the `Direct` access mode. These parameters are required when `mode` is set to `Direct`.

    Examples:

    ```
    direct:
      imagesRepo: registry.deckhouse.io/deckhouse/ee
      license: DECKHOUSE_LICENSE_KEY
    ```

    ```
    direct:
      imagesRepo: my-private-registry.com/deckhouse
      username: user
      password: password
      scheme: HTTPS
      ca: |
        -----BEGIN CERTIFICATE-----
        MIIC...
        -----END CERTIFICATE-----
    ```

    - **settings.registry.direct.ca**
      string

      The root CA certificate (in PEM format) for validating the container registry's server certificate. This is necessary if the registry uses a self-signed certificate or a certificate issued by a private Certificate Authority. If not provided, the system's trusted root CAs will be used.

    - **settings.registry.direct.checkMode**

string

Mode for checking the container registry:

- `Default` — checks for the presence of all critical components in the container registry.
- `Relax` — checks only for the presence of the Deckhouse image tag in the container registry (used for switching between Deckhouse editions).

Default: `Default`

Allowed values: `Default`, `Relax`

- **settings.registry.direct.imagesRepo**
string

Required value

The address of the container registry repository where Deckhouse images are stored.

In firewalled environments, this might be a private registry address that mirrors the public one.

Default: `registry.deckhouse.io/deckhouse/ce`

Pattern: `^[0-9a-zA-Z\.\-]+(\:[0-9]{1,5})?(\/[0-9a-zA-Z\.\-\_]+)*$`

Examples:

```
imagesRepo: registry.deckhouse.io/deckhouse/ee
```

```
imagesRepo: private-registry:5000/deckhouse/ce
```

- **settings.registry.direct.license**
string

The license key for accessing the Deckhouse container registry. This key is used as a password with the username `license-token`. This field is mutually exclusive with `username`/`password`.

Example:

```
license: DECKHOUSE_LICENSE_KEY
```

- **settings.registry.direct.password**
string

The password for authenticating with the container registry. This field is required if `license` is not provided.

Example:

```
password: registry-password
```

- **settings.registry.direct.scheme**
string

The protocol to use for connecting to the registry.

Use `HTTP` only for insecure, trusted registries (e.g., in a firewalled environment). Defaults to `HTTPS` for secure communication.

Default: `HTTPS`

Allowed values: `HTTP`, `HTTPS`

Example:

```
scheme: HTTPS
```

- **settings.registry.direct.username**
string

The username for authenticating with the container registry. This field is required if `license` is not provided.

Example:

```
username: registry-user
```

- **settings.registry.mode**
  string

  The mode for accessing the container registry with Deckhouse images.

  - `Unmanaged`: The default mode. In this mode, the `registry` module does not run an internal registry. To configure it, set the parameters in the `unmanaged` section. **Important.** Changing the registry in `Unmanaged` mode will result in a full restart of all DKP components.
  - `Direct`: A mode for direct access to the registry. It uses a “virtual” registry address that is overwritten on the fly to the real registry address. This allows for registry changes without a full component restart. **Important.** The first switch from `Unmanaged` to `Direct` mode will result in a full restart of all DKP components.
  - `Proxy`: A proxy mode for the registry. It launches static pods with a caching proxy registry. This mode allows for reducing requests to the external registry by caching images. Cached data is stored on master nodes. Access to the proxy registry is done through a fixed address, similar to `Direct` mode. **Important.** Supported only on static clusters. The first switch from `Unmanaged` to `Proxy` mode will result in a full restart of all DKP components.
  - `Local`: A local registry mode. It launches static pods with a local registry. This mode allows the cluster to operate in an isolated environment. Data storage is performed on master nodes. Access to the registry is done through a fixed address, similar to `Direct` mode. **Important.** Supported only on static clusters. The first switch from `Unmanaged` to `Local` mode will result in a full restart of all DKP components.

  Default: `Unmanaged`

  Allowed values: `Unmanaged`, `Direct`, `Proxy`, `Local`

- **settings.registry.proxy**
  object

  Parameters for the `Proxy` access mode. These parameters are required when `mode` is set to `Proxy`.

  - **settings.registry.proxy.ca**
    string

    The root CA certificate (in PEM format) for validating the container registry’s server certificate. This is necessary if the registry uses a self-signed certificate or a certificate issued by a private Certificate Authority. If not provided, the system’s trusted root CAs will be used.

  - **settings.registry.proxy.checkMode**
    string

    Mode for checking the container registry:

    - `Default` — checks for the presence of all critical components in the container registry.
    - `Relax` — checks only for the presence of the Deckhouse image tag in the container registry (used for switching between Deckhouse editions).

    Default: `Default`

    Allowed values: `Default`, `Relax`

  - **settings.registry.proxy.imagesRepo**
    string

    Required value

    The address of the container registry repository where Deckhouse images are stored.

    In firewalled environments, this might be a private registry address that mirrors the public one.

    Default: `registry.deckhouse.io/deckhouse/ce`

    Pattern: `^[0-9a-zA-Z\.\-]+(\:[0-9]{1,5})?(\/[0-9a-zA-Z\.\-\_]+)*$`

    Examples:

    ```
    imagesRepo: registry.deckhouse.io/deckhouse/ee
    ```

    ```
    imagesRepo: private-registry:5000/deckhouse/ce
    ```

  - **settings.registry.proxy.license**
    string

    The license key for accessing the Deckhouse container registry. This key is used as a password with the username `license-token`. This field is mutually exclusive with `username`/`password`.

Example:

```
license: DECKHOUSE_LICENSE_KEY
```

- **settings.registry.proxy.password**
  string

  The password for authenticating with the container registry. This field is required if `license` is not provided.

  Example:

  ```
  password: registry-password
  ```

- **settings.registry.proxy.scheme**
  string

  The protocol to use for connecting to the registry.

  Use `HTTP` only for insecure, trusted registries (e.g., in a firewalled environment). Defaults to `HTTPS` for secure communication.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

  Example:

  ```
  scheme: HTTPS
  ```

- **settings.registry.proxy.ttl**
  string

  TTL determines the lifetime of cached images. Once the TTL expires, the images will be automatically removed from the cache. If not set, the TTL defaults to 7 days (168 hours).

  Supported units: `h` (hours), `m` (minutes), `s` (seconds). The minimum allowable value is 5 minutes.

  Pattern: `^(?:(\d+)h)?(?:(\d+)m)?(?:(\d+)s)?$`

- **settings.registry.proxy.username**
  string

  The username for authenticating with the container registry. This field is required if `license` is not provided.

  Example:

  ```
  username: registry-user
  ```

- **settings.registry.unmanaged**
  object

  Parameters for the `Unmanaged` access mode.

  Examples:

  ```
  unmanaged:
    imagesRepo: registry.deckhouse.io/deckhouse/ee
    license: DECKHOUSE_LICENSE_KEY
  ```

  ```
  unmanaged:
    imagesRepo: my-private-registry.com/deckhouse
    username: user
    password: password
    scheme: HTTPS
    ca: |
      -----BEGIN CERTIFICATE-----
      MIIC...
      -----END CERTIFICATE-----
  ```

  - **settings.registry.unmanaged.ca**
    string

    The root CA certificate (in PEM format) for validating the container registry’s server certificate. This is necessary if the registry uses a self-signed certificate or a certificate

issued by a private Certificate Authority. If not provided, the system’s trusted root CAs will be used.

- **settings.registry.unmanaged.checkMode**
  string

  Mode for checking the container registry:

  - `Default` — checks for the presence of all critical components in the container registry.
  - `Relax` — checks only for the presence of the Deckhouse image tag in the container registry (used for switching between Deckhouse editions).

  Default: `Default`

  Allowed values: `Default`, `Relax`

- **settings.registry.unmanaged.imagesRepo**
  string

  Required value

  The address of the container registry repository where Deckhouse images are stored.

  In firewalled environments, this might be a private registry address that mirrors the public one.

  Default: `registry.deckhouse.io/deckhouse/ce`

  Pattern: `^[0-9a-zA-Z\.\-]+(\:[0-9]{1,5})?(\/[0-9a-zA-Z\.\-\_]+)*$`

  Examples:

  ```
  imagesRepo: registry.deckhouse.io/deckhouse/ee
  ```

  ```
  imagesRepo: private-registry:5000/deckhouse/ce
  ```

- **settings.registry.unmanaged.license**
  string

  The license key for accessing the Deckhouse container registry. This key is used as a password with the username `license-token`. This field is mutually exclusive with `username`/`password`.

  Example:

  ```
  license: DECKHOUSE_LICENSE_KEY
  ```

- **settings.registry.unmanaged.password**
  string

  The password for authenticating with the container registry. This field is required if `license` is not provided.

  Example:

  ```
  password: registry-password
  ```

- **settings.registry.unmanaged.scheme**
  string

  The protocol to use for connecting to the registry.

  Use `HTTP` only for insecure, trusted registries (e.g., in a firewalled environment). Defaults to `HTTPS` for secure communication.

  Default: `HTTPS`

  Allowed values: `HTTP`, `HTTPS`

  Example:

  ```
  scheme: HTTPS
  ```

- **settings.registry.unmanaged.username**
  string

  The username for authenticating with the container registry. This field is required if `license` is not provided.

Example:

```
username: registry-user
```

- **settings.releaseChannel**
  string

  Target Deckhouse release channel (Deckhouse will switch to it when such an opportunity appears).

  The order in which the stability of the release channel increases (from **less** stable to **more** stable): `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`, `LTS`.

  Allowed values: `Alpha`, `Beta`, `EarlyAccess`, `Stable`, `RockSolid`, `LTS`

  Example:

  ```
  releaseChannel: Stable
  ```

- **settings.reservedPublicHosts**
  object

  Settings of the reservation that keeps the hostnames of the Deckhouse web interfaces from being claimed by a workload.

  Accepted so that a ModuleConfig that already has these fields continues to validate. Admission is disabled in this patch: the policies reserved every hostname some `publicDomainTemplate` values can produce, which blocked tenant Ingresses. The settings below are ignored until a release that enforces them again. The default `mode` is `List` so that restore does not come back as `Template` for a cluster that never set the field.

  Deckhouse renders those hostnames from the `global.modules.publicDomainTemplate` global parameter and reserves them cluster-wide: an `Ingress`, or a `Gateway`, `GRPCRoute`, `HTTPRoute`, `ListenerSet` or `TLSRoute` of the Gateway API, created outside a system namespace cannot name one of them. These parameters adjust the reserved set in both directions.

  The reservation is per hostname and applies to every namespace. To exempt a whole namespace from the reservation instead, label the namespace with `security.deckhouse.io/reserved-hosts-bypass: "true"`.

  Example:

  ```
  reservedPublicHosts:
    mode: List
    additionalHosts:
    - admin.example.com
    excludedServices:
    - grafana
  ```

  - **settings.reservedPublicHosts.additionalHosts**
    array of strings

    Hostnames to reserve on top of the ones Deckhouse publishes.

    Every entry is a complete hostname, matched exactly and case-insensitively, the same way the built-in ones are. Wildcards are not supported.

    An entry always reserves, whatever `excludedServices` says, so the two settings cannot contradict each other.

    Default: `[]`

    Example:

    ```
    additionalHosts:
    - admin.example.com
    ```

    - **Element of the array**
      string

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  - **settings.reservedPublicHosts.excludedServices**
    array of strings

    Service names whose hostname must not be reserved.

    Use it for a cluster where a workload outside a system namespace legitimately serves a path under one of those hostnames.

A name is the part `global.modules.publicDomainTemplate` substitutes, `grafana` for `grafana.example.com`, not the hostname itself. The hostname it renders to is published in the `excludedHosts` key of the `d8-reserved-public-hosts` ConfigMap in the `d8-system` namespace, which is where to check that the name did what was meant.

Under `mode: Template` any name works, because the reservation covers the whole domain and not only what Deckhouse publishes. Under `mode: List` only a name Deckhouse actually publishes has an effect, because only those are reserved there; the rest are listed in the `unknownExcludedServices` key of the same ConfigMap, which is where to look if a hostname stayed reserved unexpectedly.

Excluding a name lifts the reservation of that hostname in every namespace, including the ones Deckhouse itself uses to serve it.

Default: `[]`

Example:

```
excludedServices:
- grafana
```

- **Element of the array**
  string

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **settings.reservedPublicHosts.mode**
  string

  How much of the platform's domain is reserved.

  Default is `List`.

  - `Template` — every hostname `global.modules.publicDomainTemplate` can render, plus its wildcard form where the template's `%s` is the whole first label. A module never has to ask to be covered, and a hostname is reserved as soon as the template can produce it, whether or not any Deckhouse module publishes it today. A hostname that a workload legitimately serves is given back with `excludedServices` below, or with the per-namespace label. The wildcard form is the exception: it is reserved by exact match and `excludedServices` cannot free it, so a workload that serves it needs the per-namespace label or `List` below.
  - `List` — only the hostnames of the Deckhouse services Deckhouse knows it publishes. Narrower, and it leaves a public domain that Deckhouse learns to publish later unreserved until the list catches up.

  The hostnames workloads already serve are recorded once, when `Template` first takes effect, and stay allowed. Switching from `List` to `Template` later takes that record then, so a workload that appeared while `List` was in force is grandfathered too.

  The record is taken once and is not taken again. Switching to `List` and back to `Template` keeps the record of the first time, so a workload that claimed a reserved hostname in between is denied on its next write. Recording again on every return would make the reservation avoidable by switching to `List`, claiming the hostname and switching back.

  Ecosystem applications are unaffected either way: `global.applications.publicDomainTemplate` puts the application name and its namespace in two different labels, so an application hostname has one label more than anything this reservation covers.

  Default: `List`

  Allowed values: `Template`, `List`

  Examples:

  ```
  mode: Template
  ```

  ```
  mode: List
  ```

- **settings.tolerations**
  array of objects

  The same as in the Pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, `tolerations` will be determined automatically.

  Caution! Deckhouse will stop working if `tolerations` specified are incorrect. You need to change the values to the correct ones in `ModuleConfig/deckhouse` and `deployment/deckhouse` to get Deckhouse back on track.

- **settings.tolerations.effect**
  string
- **settings.tolerations.key**
  string
- **settings.tolerations.operator**
  string
- **settings.tolerations.tolerationSeconds**
  integer
- **settings.tolerations.value**
  string

- **settings.update**
  object

  Settings of the Deckhouse update mode and windows.

  Example:

  ```
  update:
    windows:
    - from: '8:00'
      to: '15:00'
      days:
      - Tue
      - Sat
    disruptionApprovalMode: Manual
    notification:
      webhook: https://release-webhook.mydomain.com
      minimalNotificationTime: 6h
      auth:
        basic:
          username: user
          password: password
  ```

  - **settings.update.disruptionApprovalMode**
    Deprecated
    string

    Update mode for disruptive Deckhouse releases:

    - `Auto` — a disruptive release is approved automatically.
    - `Manual` — requires a manual release confirmation (set the `release.deckhouse.io/disruption-approved=true` annotation on the appropriate DeckhouseRelease resource to apply the update).

    Default: `Auto`

    Allowed values: `Auto`, `Manual`

  - **settings.update.mode**
    string

    Update mode of Deckhouse on the selected release channel.

    - `AutoPatch` — automatic update mode for patch releases.

      To change a minor version (for example, from `v1.69.*` to `v1.70.*`), confirmation is required.

      A patch version update (for example, from `v1.70.1` to `v1.70.2`) is applied taking into account the update windows, if they are set.

    - `Auto` — automatic update mode for all versions.

      A minor version updates (for example, from `v1.69.*` to `v1.70.*`) and patch version updates (for example, from `v1.70.1` to `v1.70.2`) are applied taking into account the update windows, if they are set.

    - `Manual` — manual update mode for all versions.

      Confirmation is required for updating both minor and patch versions.

    To confirm the version update, it is necessary to set the `approved` field to `true` in the corresponding resource DeckhouseRelease.

    Default: `AutoPatch`

    Allowed values: `AutoPatch`, `Auto`, `Manual`

  - **settings.update.notification**
    object

Settings for notifications of scheduled Deckhouse updates.

Has the effect **only** when the automatic update mode is set.

Example:

```
notification:
  webhook: https://release-webhook.mydomain.com
  minimalNotificationTime: 8h
  retryMinTime: 2s
```

- **settings.update.notification.auth**
  object

  Authentication settings for the webhook.

  If the parameter is omitted, the webhook will be called without authentication.

  - **settings.update.notification.auth.basic**
    object

    Basic authentication settings for the webhook.

    If the parameter is omitted, the webhook will be called without authentication.

    - **settings.update.notification.auth.basic.password**
      string

      Required value

      The password for the webhook.

      The username and password will be sent in the `Authorization` header in the format `Basic <base64(username:password)>`.

    - **settings.update.notification.auth.basic.username**
      string

      Required value

      The username for the webhook.

      The username and password will be sent in the `Authorization` header in the format `Basic <base64(username:password)>`.

  - **settings.update.notification.auth.bearerToken**
    string

    The token for the webhook.

    The token will be sent in the `Authorization` header in the format `Bearer <token>`.

- **settings.update.notification.minimalNotificationTime**
  string

  The minimum time that must pass before updating from the moment a new minor version appears on the release channel used.

  It is specified as a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

  The update mechanism ensures that Deckhouse will not be updated until a specified period of time has passed.

  When using update windows, the Deckhouse update will happen at the nearest possible update window but not before the time specified in `minimalNotificationTime` expires.

  Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

  Example:

  ```
  minimalNotificationTime: 6h
  ```

- **settings.update.notification.releaseType**
  string

  Defines the type of version for which the notification will be sent:

  - `Minor` — only for updating the minor version.
  - `All` — for any updates, including updating a patch version.

  Default: `Minor`

Allowed values: `All`, `Minor`

Example:

```
releaseType: All
```

- **settings.update.notification.retryMinTime**
  string

  The minimum time between webhook retry attempts when the webhook request fails.

  It is specified as a string containing the time unit in hours, minutes, or seconds: 30s, 1m, 2m30s, 1h, 2h30m.

  The retry mechanism uses exponential backoff: each subsequent retry attempt waits twice as long as the previous one. For example, if `retryMinTime` is set to `2s`, the retry delays will be: 2s, 4s, 8s, 16s, 32s.

  Default value is 2 seconds if not specified.

  Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m|[0-9]+s)$`

  Example:

  ```
  retryMinTime: 2s
  ```

- **settings.update.notification.tlsSkipVerify**
  boolean

  Skip TLS certificate verification while webhook request.

  Default: `false`

- **settings.update.notification.webhook**
  string

  URL for an external webhook handler.

  The POST request will be sent to the webhook URL after a new minor version of Deckhouse appears on the release channel before it is applied to the cluster.

  > **i** **Caution!** If you specify an invalid webhook address, Deckhouse update will be blocked.

  > **i** Use the minimalNotificationTime parameter if necessary to set the minimum time that must pass before updating from the moment a new minor version appears on the release channel used.

  Example of the POST request payload (`Content-Type: application/json`):

  ```
  {
    "subject":"Deckhouse",
    "version": "1.36.0",
    "requirements":  {"k8s": "1.20.0"},
    "changelogLink":
  "https://github.com/deckhouse/deckhouse/changelog/1.36.md",
    "applyTime": "2023-01-01T14:30:00Z00:00",
    "message": "New Deckhouse Release 1.36.0 is available. Release will be
  applied at: Friday, 01-Jan-23 14:30:00 UTC"
  }
  ```

  Description of POST request fields:

  - `subject` — string, the update event type. Possible values: `Deckhouse`, `Module`;
  - `moduleName` — string, the name of the module. Set only if `subject: Module`.
  - `version` - string, x.y.z (semantic versioning);
  - `requirements` - object, version requirements;
  - `changelogLink` - string, a URL to the minor version changelog;
  - `applyTime` - string, date and time of the scheduled update (taking into account the configured update windows) in RFC3339 format;
  - `message` - string, a text message about the availability of the new minor version and the scheduled update time.

  Pattern: `^https?://[^\s/$.?#].[^\s]*$`

Example:

```
webhook: https://webhook.site/#!/bc8f71ac-c182-4181-9159-6ba6950afffa
```

- **settings.update.windows**
  array of objects

  List of update windows during the day.

  - **settings.update.windows.days**
    array of strings

    The days of the week on which the update window is applied.

    Example:

    ```
    days:
    - Mon
    - Wed
    ```

    - **Element of the array**
      string

      Day of the week.

      Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

      Example:

      ```
      Mon
      ```

  - **settings.update.windows.from**
    string

    Required value

    Start time of the update window (UTC timezone).

    Should be less than the end time of the update window.

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    from: '13:00'
    ```

  - **settings.update.windows.to**
    string

    Required value

    End time of the update window (UTC timezone).

    Should be more than the start time of the update window.

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    to: '18:30'
    ```

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

ConversionWebhook

Short names: cwhc

Scope: Cluster

Version: v1alpha1

ConversionWebhook is the Schema for the conversionwebhooks API

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **context**
  array of objects

  Run a hook on a Kubernetes object changes.

  - **context.kubernetes**
    object
    - **context.kubernetes.allowFailure**
      boolean

      If `true`, Shell-operator skips the hook execution errors. If `false` or the parameter is not set, the hook is restarted after a 5 seconds delay in case of an error.

    - **context.kubernetes.apiVersion**
      string

      Is an optional group and version of object API. For example, it is `v1` for core objects (Pod, etc.), `rbac.authorization.k8s.io/v1beta1` for ClusterRole and `monitoring.coreos.com/v1` for prometheus-operator.

    - **context.kubernetes.fieldSelector**
      object
      - **context.kubernetes.fieldSelector.matchExpressions**
        array of objects

        Required value

        - **context.kubernetes.fieldSelector.matchExpressions.field**
          string

          Required value

        - **context.kubernetes.fieldSelector.matchExpressions.operator**
          string

          Required value

        - **context.kubernetes.fieldSelector.matchExpressions.value**
          string
    - **context.kubernetes.includeSnapshotsFrom**
      array of strings

      An array of names of kubernetes bindings in a hook. When specified, a list of monitored objects from that bindings will be added to the binding context in a snapshots field. Self-include is also possible.

    - **context.kubernetes.jqFilter**
      string

      An optional parameter that specifies event filtering using jq syntax. The hook will be triggered on the “Modified” event only if the filter result is changed after the last event.

    - **context.kubernetes.kind**
      string

      Required value

      Is the type of a monitored Kubernetes resource. This field is required.

    - **context.kubernetes.labelSelector**
      object

      A label selector is a label query over a set of resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.

      - **context.kubernetes.labelSelector.matchExpressions**
        array of objects

        matchExpressions is a list of label selector requirements. The requirements are ANDed.

        A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

- **context.kubernetes.labelSelector.matchExpressions.key**
  string

  Required value

  key is the label key that the selector applies to.

- **context.kubernetes.labelSelector.matchExpressions.operator**
  string

  Required value

  operator represents a key’s relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

- **context.kubernetes.labelSelector.matchExpressions.values**
  array of strings

  values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

- **context.kubernetes.labelSelector.matchLabels**
  object

  matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are ANDed.

- **context.kubernetes.nameSelector**
  object
  - **context.kubernetes.nameSelector.matchNames**
    array of strings

    Required value

- **context.kubernetes.namespace**
  object

  Filters to choose namespaces.

  - **context.kubernetes.namespace.labelSelector**
    object

    A label selector is a label query over a set of resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.

    - **context.kubernetes.namespace.labelSelector.matchExpressions**
      array of objects

      matchExpressions is a list of label selector requirements. The requirements are ANDed.

      A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

      - **context.kubernetes.namespace.labelSelector.matchExpressions.key**
        string

        Required value

        key is the label key that the selector applies to.

      - **context.kubernetes.namespace.labelSelector.matchExpressions.operator**
        string

        Required value

        operator represents a key’s relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

      - **context.kubernetes.namespace.labelSelector.matchExpressions.values**
        array of strings

        values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

    - **context.kubernetes.namespace.labelSelector.matchLabels**
      object

      matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are

ANDed.

- **context.kubernetes.namespace.nameSelector**
  object
  - **context.kubernetes.namespace.nameSelector.matchNames**
    array of strings

    Required value

- **context.kubernetes.queue**
  string

  A name of a separate queue. It can be used to execute long-running hooks in parallel with hooks in the “main” queue.

- **context.name**
  string

  Required value

  It is used to distinguish different bindings during runtime.

- **conversions**
  array of objects

  Required value

  This binding transforms a hook into a handler for conversions defined in CustomResourceDefinition. The Shell-operator updates a CRD with `.spec.conversion`, starts HTTPS server, and runs hooks to handle ConversionReview requests.

  - **conversions.from**
    string

    Required value

    A version of a custom resource that hook can convert.

  - **conversions.handler**
    object

    Required value

    Code of the conversion handler.

    - **conversions.handler.python**
      string

      Python script handler for object.

  - **conversions.includeSnapshotsFrom**
    array of strings

    An array of names of Kubernetes bindings in a hook. When specified, a list of monitored objects from these bindings will be added to the binding context in the snapshots field.

  - **conversions.to**
    string

    Required value

    A version of a custom resource that hook can produce.

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **status**
  object

  Status defines the observed state of ConversionWebhook.

ValidationWebhook

Short names: vwhc

Scope: Cluster
Version: v1alpha1

ValidationWebhook is the Schema for the validationwebhooks API

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **context**
  array of objects

  Run a hook on a Kubernetes object changes.

  - **context.kubernetes**
    object
    - **context.kubernetes.allowFailure**
      boolean

      If `true`, Shell-operator skips the hook execution errors. If `false` or the parameter is not set, the hook is restarted after a 5 seconds delay in case of an error.

    - **context.kubernetes.apiVersion**
      string

      Is an optional group and version of object API. For example, it is `v1` for core objects (Pod, etc.), `rbac.authorization.k8s.io/v1beta1` for ClusterRole and `monitoring.coreos.com/v1` for prometheus-operator.

    - **context.kubernetes.fieldSelector**
      object
      - **context.kubernetes.fieldSelector.matchExpressions**
        array of objects

        Required value

        - **context.kubernetes.fieldSelector.matchExpressions.field**
          string

          Required value

        - **context.kubernetes.fieldSelector.matchExpressions.operator**
          string

          Required value

        - **context.kubernetes.fieldSelector.matchExpressions.value**
          string
    - **context.kubernetes.includeSnapshotsFrom**
      array of strings

      An array of names of kubernetes bindings in a hook. When specified, a list of monitored objects from that bindings will be added to the binding context in a snapshots field. Self-include is also possible.

    - **context.kubernetes.jqFilter**
      string

      An optional parameter that specifies event filtering using jq syntax. The hook will be triggered on the “Modified” event only if the filter result is changed after the last event.

    - **context.kubernetes.kind**
      string

      Required value

      Is the type of a monitored Kubernetes resource. This field is required.

    - **context.kubernetes.labelSelector**
      object

      A label selector is a label query over a set of resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.

      - **context.kubernetes.labelSelector.matchExpressions**
        array of objects

        matchExpressions is a list of label selector requirements. The requirements are ANDed.

        A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

        - **context.kubernetes.labelSelector.matchExpressions.key**
          string

          Required value

key is the label key that the selector applies to.

- **context.kubernetes.labelSelector.matchExpressions.operator**
  string

  Required value

  operator represents a key’s relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

- **context.kubernetes.labelSelector.matchExpressions.values**
  array of strings

  values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

- **context.kubernetes.labelSelector.matchLabels**
  object

  matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are ANDed.

- **context.kubernetes.nameSelector**
  object
  - **context.kubernetes.nameSelector.matchNames**
    array of strings

    Required value

- **context.kubernetes.namespace**
  object

  Filters to choose namespaces.

  - **context.kubernetes.namespace.labelSelector**
    object

    A label selector is a label query over a set of resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.

    - **context.kubernetes.namespace.labelSelector.matchExpressions**
      array of objects

      matchExpressions is a list of label selector requirements. The requirements are ANDed.

      A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

      - **context.kubernetes.namespace.labelSelector.matchExpressions.key**
        string

        Required value

        key is the label key that the selector applies to.

      - **context.kubernetes.namespace.labelSelector.matchExpressions.operator**
        string

        Required value

        operator represents a key’s relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

      - **context.kubernetes.namespace.labelSelector.matchExpressions.values**
        array of strings

        values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

    - **context.kubernetes.namespace.labelSelector.matchLabels**
      object

      matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are ANDed.

  - **context.kubernetes.namespace.nameSelector**
    object
    - **context.kubernetes.namespace.nameSelector.matchNames**

array of strings

Required value

- **context.kubernetes.queue**
  string

  A name of a separate queue. It can be used to execute long-running hooks in parallel with hooks in the “main” queue.

- **context.name**
  string

  Required value

  It is used to distinguish different bindings during runtime.

- **handler**
  object

  Required value

  Code of the ValidatingWebhook handler

  - **handler.cel**
    string

    CEL rules handler for object.

  - **handler.python**
    string

    Python script handler for object.

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **status**
  object

  Status defines the observed state of ValidationWebhook.

- **validationObject**
  object

  Required value

  ValidatingWebhook describes an webhook and the resources and operations it applies to.

  - **validationObject.failurePolicy**
    string

    Defines how errors from the hook are handled.

  - **validationObject.group**
    string

    A key to include snapshots from a group of schedule and Kubernetes bindings. See grouping.

  - **validationObject.labelSelector**
    object

    A label selector is a label query over a set of resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.

    - **validationObject.labelSelector.matchExpressions**
      array of objects

      matchExpressions is a list of label selector requirements. The requirements are ANDed.

      A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

      - **validationObject.labelSelector.matchExpressions.key**
        string

        Required value

        key is the label key that the selector applies to.

        - **validationObject.labelSelector.matchExpressions.operator**
          string

          Required value

          operator represents a key’s relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

        - **validationObject.labelSelector.matchExpressions.values**
          array of strings

          values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

      - **validationObject.labelSelector.matchLabels**
        object

        matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are ANDed.

  - **validationObject.matchConditions**
    array of objects

    An optional list of match conditions for fine-grained request filtering. Available only since v1.27 of Kubernetes.

    MatchCondition represents a condition which must by fulfilled for a request to be sent to a webhook.

      - **validationObject.matchConditions.expression**
        string

        Required value

        Expression represents the expression which will be evaluated by CEL. Must evaluate to bool. CEL expressions have access to the contents of the AdmissionRequest and Authorizer, organized into CEL variables:

        ‘object’ - The object from the incoming request. The value is null for DELETE requests. ‘oldObject’ - The existing object. The value is null for CREATE requests. ‘request’ - Attributes of the admission request(/pkg/apis/admission/types.go#AdmissionRequest). ‘authorizer’ - A CEL Authorizer. May be used to perform authorization checks for the principal (user or service account) of the request. See https://pkg.go.dev/k8s.io/apiserver/pkg/cel/library#Authz ‘authorizer.requestResource’ - A CEL ResourceCheck constructed from the ‘authorizer’ and configured with the request resource. Documentation on CEL: https://kubernetes.io/docs/reference/using-api/cel/

        Required.

      - **validationObject.matchConditions.name**
        string

        Required value

        Name is an identifier for this match condition, used for strategic merging of MatchConditions, as well as providing an identifier for logging purposes. A good name should be descriptive of the associated expression. Name must be a qualified name consisting of alphanumeric characters, ‘-‘, ‘*’ or ‘.’, and must start and end with an alphanumeric character (e.g. ‘MyName’, or ‘my.name’, or ‘123-abc’, regex used for validation is ‘([A-Za-z0-9][-A-Za-z0-9.]*)?[A-Za-z0-9]’) with an optional DNS subdomain prefix and ‘/’ (e.g. ‘example.com/MyName’)

        Required.

  - **validationObject.name**
    string

    Required value

    Name should be a domain with at least three segments separated by dots.

  - **validationObject.namespace**
    object

    Namespace filtering. Only labelSelector is supported (not nameSelector). To filter by namespace name, use labelSelector with kubernetes.io/metadata.name label.

      - **validationObject.namespace.labelSelector**
        object

        labelSelector is a selector to filter namespaces by labels. Use kubernetes.io/metadata.name label to filter by namespace name.

          - **validationObject.namespace.labelSelector.matchExpressions**

array of objects

matchExpressions is a list of label selector requirements. The requirements are ANDed.

A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

- **validationObject.namespace.labelSelector.matchExpressions.key**
  string

  Required value

  key is the label key that the selector applies to.

- **validationObject.namespace.labelSelector.matchExpressions.operator**
  string

  Required value

  operator represents a key's relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

- **validationObject.namespace.labelSelector.matchExpressions.values**
  array of strings

  values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

- **validationObject.namespace.labelSelector.matchLabels**
  object

  matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is "key", the operator is "In", and the values array contains only "value". The requirements are ANDed.

- **validationObject.rules**
  array of objects

  A required list of rules used to determine if a request to the Kubernetes API server should be sent to the hook.

  RuleWithOperations is a tuple of Operations and Resources. It is recommended to make sure that all the tuple expansions are valid.

  - **validationObject.rules.apiGroups**
    array of strings

    APIGroups is the API groups the resources belong to. ' *' is all groups. If ''* is present, the length of the slice must be one. Required.

  - **validationObject.rules.apiVersions**
    array of strings

    APIVersions is the API versions the resources belong to. ' *' is all versions. If ''* is present, the length of the slice must be one. Required.

  - **validationObject.rules.operations**
    array of strings

    Operations is the operations the admission hook cares about - CREATE, UPDATE, DELETE, CONNECT or * for all of those operations and any future admission operations that are added. If '*' is present, the length of the slice must be one. Required.

    - **Element of the array**
      string

      OperationType specifies an operation for a request.

  - **validationObject.rules.resources**
    array of strings

    Resources is a list of resources this rule applies to.

    For example: 'pods' means pods. 'pods/log' means the log subresource of pods. ' *' means all resources, but not subresources. 'pods/'* means all subresources of pods. '*/scale' means all scale subresources. '*/*' means all resources and their subresources.

    If wildcard is present, the validation rule will ensure resources do not overlap with each other.

    Depending on the enclosing object, subresources might not be allowed. Required.

  - **validationObject.rules.scope**
    string

scope specifies the scope of this rule. Valid values are “Cluster”, “Namespaced”, and “ ” *“Cluster” means that only cluster-scoped resources will match this rule. Namespace API objects are cluster-scoped. “Namespaced” means that only namespaced resources will match this rule. “”* means that there are no scope restrictions. Subresources match the scope of their parent resource. Default is “*”.

- **validationObject.sideEffects**
  string

  Determines whether the hook is dryRun-aware.

- **validationObject.timeoutSeconds**
  integer

  Seconds API server should wait for a hook to respond before treating the call as a failure. Default is 10 (seconds).

Usage

> **i** The module lifecycle stage: General Availability

Setting up the update mode

You can manage DKP updates in the following ways:

- Using the `settings.update` parameter of the ModuleConfig `deckhouse` resource.
- Using the `disruptions` parameters section of the NodeGroup resource.

Update windows configuration

You can configure the time windows when Deckhouse will automatically install updates in the following ways:

- For general update management, in the `update.windows` parameter of the `deckhouse` ModuleConfig resource.
- For managing disruptive updates, in the `disruptions.automatic.windows` and `disruptions.rollingUpdate.windows` parameters of the NodeGroup resource.

An example of setting up two daily update windows — from 8 a.m. to 10 a.m. and from 8 p.m. to 10 p.m. (UTC):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: EarlyAccess
    update:
      windows:
        - from: "8:00"
          to: "10:00"
        - from: "20:00"
          to: "22:00"
```

You can also set up updates on certain days, for example, on Tuesdays and Saturdays from 6 p.m. to 7:30 p.m. (UTC):

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    releaseChannel: Stable
    update:
      windows:
        - from: "18:00"
          to: "19:30"
          days:
            - Tue
            - Sat
```

Manual update confirmation

Manual confirmation of Deckhouse version updates is provided in the following cases:

- The Deckhouse update confirmation mode is enabled.

  This means that the parameter settings.update.mode in the ModuleConfig `deckhouse` is set to `Manual` (confirmation for both patch and minor versions of Deckhouse) or `AutoPatch` (confirmation for the minor version of Deckhouse). Run the following command to confirm the update (use the corresponding Deckhouse version):

  ```
  d8 k patch DeckhouseRelease <VERSION> --type=merge -p='{"approved": true}'
  ```

- If automatic application of disruptive updates is disabled for a node group.

  This means that the corresponding NodeGroup has the parameter `spec.disruptions.approvalMode` set to `Manual`.

  For updating **each** node in such a group, the node must have `update.node.deckhouse.io/disruption-approved=` annotation. Example:

  ```
  d8 k annotate node ${NODE_1} update.node.deckhouse.io/disruption-approved=
  ```

Deckhouse update notifications

In the `Auto` update mode, you can configure a webhook call to receive a notification about an upcoming minor Deckhouse version update.

In addition, notifications are generated not only for Deckhouse updates but also for updates of any modules, including their individual updates. In some cases, the system may initiate the sending of multiple notifications at once (10–20 notifications) at approximately 15-second intervals.

> **i** Notifications are available only in the `Auto` update mode; in the `Manual` mode they are not generated.

> **i** Specifying a webhook is optional: if the `update.notification.webhook` parameter is not set but the `update.notification.minimalNotificationTime` parameter is specified, the update will still be postponed for the specified period. In this case, the appearance of a DeckhouseRelease resource in the cluster with the name of the new version can be considered the notification of its availability.

Notifications are sent only once for a specific update. If something goes wrong (for example, the webhook receives incorrect data), they will not be resent automatically. To resend the notification, you must delete the corresponding DeckhouseRelease resource.

Example of notification configuration:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    update:
      releaseChannel: Stable
      mode: Auto
      notification:
        webhook: https://release-webhook.mydomain.com
```

After a new minor Deckhouse version appears on the selected release channel, but before it is applied in the cluster, a POST request will be sent to the configured webhook address.

The minimalNotificationTime parameter allows you to postpone the update installation for the specified period, providing time to react to the notification while respecting update windows. If the webhook is unavailable, each failed attempt to send the notification will postpone the update by the same duration, which may lead to the update being deferred indefinitely.

> ⚠ If your webhook returns any status code out of 2xx range, DKP retries sending the notification up to five times with exponential backoff. If all attempts fail, the release is blocked until the webhook becomes available again.

For easier error handling and debugging, when returning error codes the webhook should return a JSON response with the following structure:

- `code` — optional internal error code for programmatic handling;
- `message` — a human-readable description of what went wrong.

If the webhook returns a successful HTTP status (2xx), DKP treats the notification as successful regardless of the response body.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    update:
      releaseChannel: Stable
      mode: Auto
      notification:
        webhook: https://release-webhook.mydomain.com
        minimalNotificationTime: 8h
```

Minimal Go webhook example...

```
package main

import (
  "encoding/json"
  "fmt"
  "log"
  "net/http"
)

// Payload structure Deckhouse sends in POST body.
type WebhookData struct {
  Subject      string          `json:"subject"`
  Version      string          `json:"version"`
  Requirements  map[string]string `json:"requirements,omitempty"`
  ChangelogLink string          `json:"changelogLink,omitempty"`
  ApplyTime    string          `json:"applyTime,omitempty"`
  Message      string          `json:"message"`
```

```
}

// Response structure that Deckhouse expects from webhook on error
type ResponseError struct {
  Code    string `json:"code,omitempty"`
  Message string `json:"message"`
}

func handler(w http.ResponseWriter, r *http.Request) {
  if r.Method != http.MethodPost {
    w.WriteHeader(http.StatusMethodNotAllowed)
    return
  }
  defer r.Body.Close()

  var data WebhookData
  if err := json.NewDecoder(r.Body).Decode(&data); err != nil {
    log.Printf("failed to decode payload: %v", err)
    w.WriteHeader(http.StatusInternalServerError)
    return
  }

  // Print payload fields
  log.Printf("subject=%s version=%s applyTime=%s changelog=%s requirements=%v",
    data.Subject, data.Version, data.ApplyTime, data.ChangelogLink, data.Requirements)
  log.Printf("message=%s", data.Message)

  // Example conditional logic: fail intentionally for testing
  if data.Version == "v0.0.0-fail" {
    // Return structured error response with error status code
    errorResp := ResponseError{
      Code:    "TEST_FAILURE",
      Message: "intentional failure for testing",
    }

    w.WriteHeader(http.StatusBadRequest)
    json.NewEncoder(w).Encode(errorResp)
    return
  }

  // Return success response with 2xx status code
  w.WriteHeader(http.StatusOK)
  w.Write([]byte("Notification processed successfully"))
}

func main() {
  mux := http.NewServeMux()
  mux.HandleFunc("/webhook", handler)

  addr := ":8080"
  fmt.Printf("listening on %s, POST to http://localhost%s/webhook\n", addr, addr)
  if err := http.ListenAndServe(addr, mux); err != nil {
    log.Fatal(err)
  }
}
```

Reserving the hostnames of the platform web interfaces

DKP publishes its own web interfaces under hostnames rendered from the `publicDomainTemplate` global parameter: the API, the web interface, Grafana, Dex, the kubeconfig generator and others. Those hostnames are reserved cluster-wide. An Ingress, HTTPRoute, GRPCRoute, TLSRoute, ListenerSet or Gateway resource created outside a namespace labeled `heritage: deckhouse` cannot claim such a name, and such a request is rejected with a message naming the hostname:

```
Hostname console.example.com is reserved for Deckhouse platform services
and cannot be claimed outside a system namespace
```

Only resource creation and modification are checked. An object that already claims a reserved hostname keeps serving traffic until it is modified.

Reserved hostnames

The reserved set is controlled by the `settings.reservedPublicHosts.mode` parameter of the ModuleConfig `deckhouse`:

- `Template` (by default): Every hostname the template can render are reserved.
- `List`: Only the hostnames of the services DKP knows it publishes.

For details on each reservation method, refer to the parameter description.

Excluding a name from reservation

Under `Template` mode, the reservation covers not only the hostnames used by DKP but also others matching the template. A workload that requires such a hostname can be excluded from reservation in one of the following ways:

- Free a single hostname with the `settings.reservedPublicHosts.excludedServices` parameter.

  The parameter includes not the entire hostname, only the service name (the value included in the template). For example, for the `%s.example.com` template, the `grafana` value excludes `grafana.example.com` from reservation.

- Exclude a whole namespace with the `security.deckhouse.io/reserved-hosts-bypass: "true"` label.

  The label is set on the namespace, not on the object, so a tenant who is allowed to manage resources only in their own namespaces can’t set it.

- Return to the previous reservation by setting `mode: List`.

The following is an example that frees `grafana.example.com` from reservation and additionally reserves `admin.corp.example.org` that doesn’t match the template:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse
spec:
  version: 1
  settings:
    reservedPublicHosts:
      mode: Template
      excludedServices:
        - grafana
      additionalHosts:
        - admin.corp.example.org
```

The `settings.reservedPublicHosts.additionalHosts` parameter lets you reserve additional hostnames regardless of the `excludedServices` value.

⚠ The `excludedServices` parameter accepts service names only, therefore it can’t be used to exclude a wildcard name from reservation.

To allow a workload to use a wildcard name (such as `*.example.com`), disable the checking for its namespace using the above-mentioned label or use the `List` mode.

Excluding the shared Gateway

The Gateway to which DKP module ListenerSet resources are attached is automatically excluded from reservation, regardless of the hostnames specified in its listeners.

The following Gateway is considered the shared Gateway:

- The Gateway specified in the global `gatewayAPIGateway` parameter.
- If the parameter is not set, the Gateway specified in the `default-gateway` ConfigMap in the `d8-alb` namespace.

The exclusion applies by name and namespace and only to the Gateway resource. It does not apply to other resources using the same hostnames, such as Ingress, HTTPRoute, or another Gateway.

This exclusion is necessary because such a Gateway typically has a listener with a wildcard hostname and wildcard certificate and is often located in a dedicated namespace without the `heritage: deckhouse` label. Without the exclusion, the Gateway could not be modified after hostname reservation is enabled, even though the attached DKP routes continue to depend on it.

Any other Gateway with a `*.example.com` listener is rejected, even if that hostname was already in use when

reservation was enabled. To use such a Gateway, add the `security.deckhouse.io/reserved-hosts-bypass: "true"` label to its namespace or enable `List` mode.

If the shared Gateway is neither specified in the `gatewayAPIGateway` parameter nor defined through the `default-gateway` ConfigMap, no automatic exclusion is applied.

Reservation with existing applications in the cluster

When hostname reservation takes effect, hostnames already used by applications are automatically added to the list of allowed hostnames. Otherwise, the next modification of the corresponding resources would be rejected.

The list is stored in the `grandfatheredHosts` key of the `d8-reserved-public-hosts` ConfigMap in the `d8-system` namespace, separately from hostnames manually excluded from reservation.

The list is created only when `Template` mode is enabled for the first time and is not updated afterward. This prevents reservation from being bypassed: an application cannot claim a reserved hostname after reservation has been enabled and then have that hostname allowed through subsequent regeneration of the list.

If an application that used a hostname allowed this way is removed and the hostname is no longer needed, remove the corresponding entry from `grandfatheredHosts`.

Before being stored, hostnames are converted to lowercase and any trailing dot is removed. Invalid values, such as a URL instead of a hostname, are ignored. If a resource uses the corresponding reserved hostname, its next modification will be rejected.

Wildcard hostnames, such as `*.example.com`, are not added to `grandfatheredHosts`. Otherwise, an application could use any hostname matching the template, including hostnames of services that DKP may publish in the future. To allow an application to use a wildcard hostname in the platform domain, add the `security.deckhouse.io/reserved-hosts-bypass: "true"` label to its namespace or enable `List` mode.

This restriction does not apply to the Gateway to which DKP is attached. This Gateway is automatically excluded from the check.

Switching from `Template` mode to `List` mode and back does not regenerate `grandfatheredHosts`. Therefore, hostnames that applications start using while `List` mode is enabled are not added to the list of allowed hostnames after switching back to `Template` mode. When the corresponding resources are modified, these hostnames are checked according to the standard reservation rules.

Checking reservation status

Information about the current reservation status is stored in the `d8-reserved-public-hosts` ConfigMap in the `d8-system` namespace.

To view the ConfigMap, run the following command:

```
d8 k -n d8-system get configmap d8-reserved-public-hosts -o yaml
```

The ConfigMap contains the following keys:

Key	Description
`mode`	Current reservation mode. If `publicDomainTemplate` does not match the schema, the value is `List`
`hostPattern`	Regular expression describing the hostnames that can be generated from the template. The value is empty in `List` mode
`hosts`	Hostnames matched exactly. In `Template` mode, contains the values from `additionalHosts` and the wildcard domain name; in `List` mode, contains the hostnames of platform services
`allowedHosts`	Hostnames excluded from matching against `hostPattern`: hostnames from `excludedServices` and `grandfatheredHosts`
`excludedHosts`	Hostnames excluded from reservation using `excludedServices`
`grandfatheredHosts`	Hostnames that were already used by applications when reservation was enabled and were automatically added to the list of allowed hostnames
`sharedGateway`	Gateway to which DKP is attached, in the `<NAMESPACE>/<NAME>` format. The value is empty if no such Gateway is configured or detected

Key	Description
`platformHosts`	Hostnames of services published by DKP. Used as a reference in both reservation modes
`unknownExcludedServices`	Names from `excludedServices` that are not published by any module. A value in this key most likely indicates a typo in the service name

Collect debug info

Read the FAQ to learn more about collecting debug information.

FAQ

> **i** The module lifecycle stage: General Availability

How to run kube-bench in my cluster?

First, you have to exec in Deckhouse Pod:

```
d8 k -n d8-system exec -ti svc/deckhouse-leader -c deckhouse -- bash
```

Then you have to select which node you want to run kube-bench.

- Run on random node:

  ```
  curl -s https://raw.githubusercontent.com/aquasecurity/kube-bench/main/job.yaml | d8 k create -f -
  ```

- Run on specific node, e.g. control-plane node:

  ```
  curl -s https://raw.githubusercontent.com/aquasecurity/kube-bench/main/job.yaml | d8 k apply -f - --dry-run=client -o json | jq '.spec.template.spec.tolerations=[{"operator": "Exists"}] | .spec.template.spec.nodeSelector={"node-role.kubernetes.io/control-plane": ""}' | d8 k create -f -
  ```

Then you can check report:

```
d8 k logs job.batch/kube-bench
```

> **⚠** Deckhouse set the log retention period to 7 days. However, according to the security requirements specified in kube-bench, logs should be retained for at least 30 days. Use separate storage for logs if you need to keep logs for more than 7 days.

How to collect debug info?

1. Create a diagnostic archive with the `d8` utility, redirecting its output (stdout) to a file:

   ```
   d8 system collect-debug-info > deckhouse-debug-$(date +"%Y_%m_%d").tar.gz
   ```

2. Send the resulting archive to the Deckhouse team for further debugging.

> **i** The `--exclude` flag omits the specified items from the archive. Example:

```
d8 system collect-debug-info --exclude=queue global-values > deckhouse-debug-$(date +"%Y_%m_%d").tar.gz
```

> **i** The `--list-exclude` flag prints the list of items available for exclusion. Example:

```
d8 system collect-debug-info --list-exclude
```

The following information is produced when creating the archive. Names in the “File in archive” column

correspond to top-level items inside the resulting `tar.gz` archive. Certain sensitive values (e.g., `kubeRBACProxyCA` and `registry.dockercfg`) are excluded.

Category	Collected data	File in archive
Deckhouse	Deckhouse queue state	`queue`
	Deckhouse values (excluding `kubeRBACProxyCA` and `registry.dockercfg`)	`global-values`
	Current `deckhouse` pod version	`deckhouse-version`
	All `DeckhouseRelease` objects	`deckhouse-releases`
	Deckhouse pod logs	`deckhouse-logs`
	Manifests of controllers and pods from all Deckhouse namespaces	`d8-all`
Cluster objects	`NodeGroup`	`node-groups`
	`NodeGroupConfiguration`	`node-group-configuration`
	`Node`	`nodes`
	`Machine`	`machines`
	`Instance`	`instances`
	`StaticInstance`	`staticinstances`
	`MachineDeployment`	`cloud-machine-deployment`, `static-machine-deployment`
	`ClusterAuthorizationRule`	`cluster-authorization-rules`
	`AuthorizationRule`	`authorization-rules`
	`ModuleConfig`	`module-configs`
	Events (all namespaces)	`events`
Modules and states	List of enabled modules	`deckhouse-enabled-modules`
	`ModuleSource` objects in the cluster	`deckhouse-module-sources`
	`ModulePullOverride` objects in the cluster	`deckhouse-module-pull-overrides`
	Modules in `maintenance` mode	`deckhouse-maintenance-modules`
Controller logs and manifests	`machine-controller-manager` logs	`mcm-logs`
	`cloud-controller-manager` logs	`ccm-logs`
	`csi-controller` logs	`csi-controller-logs`
	`cluster-autoscaler` logs	`cluster-autoscaler-logs`
	Vertical Pod Autoscaler admission controller logs	`vpa-admission-controller-logs`
	Vertical Pod Autoscaler recommender logs	`vpa-recommender-logs`
	Vertical Pod Autoscaler updater logs	`vpa-updater-logs`
	`capi-controller-manager` YAML	`capi-controller-manager`
	`caps-controller-manager` YAML	`caps-controller-manager`
	`machine-controller-manager` YAML	`machine-controller-manager`
Monitoring and alerts	Prometheus logs	`prometheus-logs`
	Active (firing) Prometheus alerts	`alerts`
	Pods not in `Running` (excluding `Completed` and `Evicted`)	`bad-pods`
	List of Audit Policies	`audit-policy`

Category	Collected data	File in archive
Network	All objects in the `d8-istio` namespace	`d8-istio-resources`
	All `istio` custom resources	`d8-istio-custom-resources`
	Envoy configuration for `istio`	`d8-istio-envoy-config`
	`istio` logs	`d8-istio-system-logs`
	`istio` ingress gateway logs	`d8-istio-ingress-logs`
	`istio` users logs	`d8-istio-users-logs`
	Cilium connection status (`cilium health status`)	`cilium-health-status`

How to debug pod problems with ephemeral containers?

Run the following command:

```
d8 k -n <namespace_name> debug -it <pod_name> --image=ubuntu <container_name>
```

More info in official documentation.

How to debug node problems with ephemeral containers?

Run the following command:

```
d8 k debug node/mynode -it --image=ubuntu
```

More info in official documentation.

The deckhouse-tools module

> **i** The module lifecycle stage: General Availability

> **i** Web interfaces associated with the module: deckhouse-tools

The module creates a web UI with links to download Deckhouse CLI tool for various operating systems.

The web interface address is composed based on the publicDomainTemplate template of the Deckhouse global configuration parameter (the `%s` key is replaced with `tools`).

For example, if `publicDomainTemplate` is set to `%s-kube.company.my`, the tools web interface will be exposed at `tools-kube.company.my`.

Configuration

> **i** The module lifecycle stage: General Availability

The module does not have any mandatory parameters.

An example of the configuration

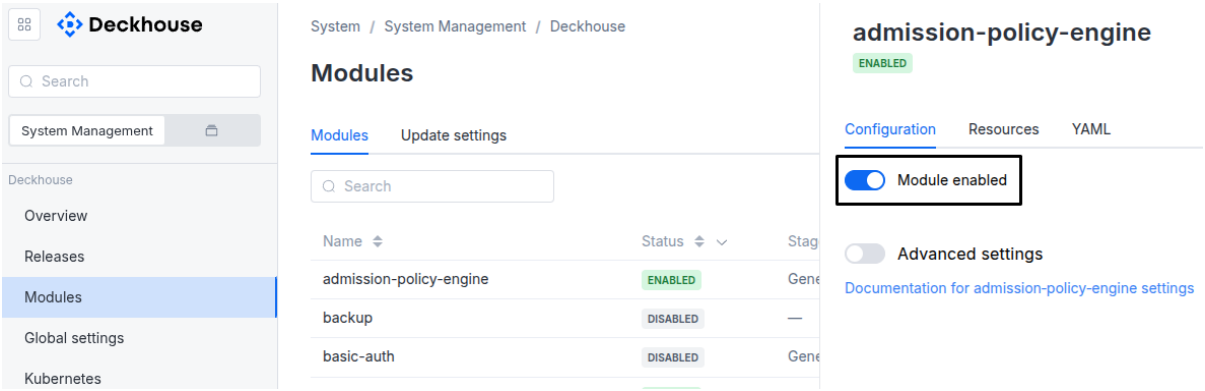
```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse-tools
spec:
  enabled: true
  version: 1
```

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `deckhouse-tools` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable deckhouse-tools
```

- **Using ModuleConfig `deckhouse-tools`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `deckhouse-tools` (create it if necessary);

  Example of a manifest to enable module `deckhouse-tools`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: deckhouse-tools
spec:
  enabled: true
```

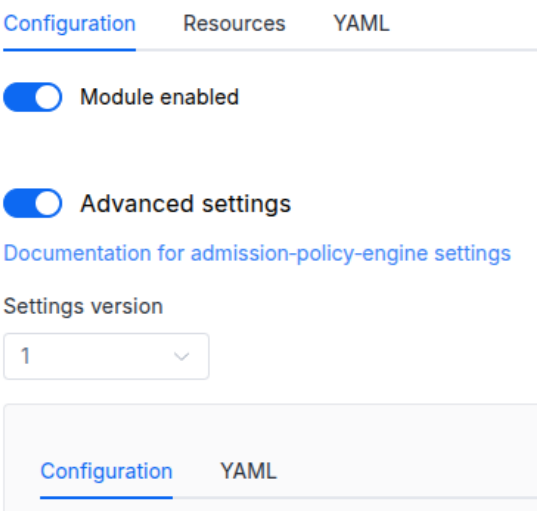
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `deckhouse-tools` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `deckhouse-tools` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `deckhouse-`

`tools`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `deckhouse-tools` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc deckhouse-tools
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `deckhouse-tools` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `deckhouse-tools`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: deckhouse-tools
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.auth**
    object

    Parameters to authenticate and authorize access to the module’s web interface.

    - **settings.auth.allowedUserEmails**
      array of strings

      An array of emails of users that can access module’s public web interfaces.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

    - **settings.auth.allowedUserGroups**
      array of strings

      An array of groups whose users can browse the module’s web interface.

      This parameter is used if the `user-authn` module is enabled or if the `externalAuthentication` parameter is set.

      > **Caution!** Note that you must add those groups to the appropriate field in the `DexProvider` config if this module is used together with the `user-authn` one.

      Default: `[]`

      Examples:

      ```
      allowedUserGroups:
      - admin
      - users
      ```

      ```
      allowedUserGroups: []
      ```

    - **settings.auth.externalAuthentication**

object

Parameters to enable external authentication based on the Ingress NGINX external-auth mechanism that uses the Nginx auth_request module.

> **i** External authentication is enabled automatically if the user-authn module is enabled.

- **settings.auth.externalAuthentication.authSignInURL**
  string

  The URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response).

  Default:

  Example:

  ```
  authSignInURL: https://$host/dex-authenticator/sign_in
  ```

- **settings.auth.externalAuthentication.authURL**
  string

  The URL of the authentication service.

  If the user is authenticated, the service should return an HTTP 200 response code.

  Default:

  Example:

  ```
  authURL: https://deckhouse-tools-dex-authenticator.d8-
  system.svc.cluster.local/dex-authenticator/auth
  ```

- **settings.https**
  object

  What certificate type to use.

  This parameter completely overrides the `global.modules.https` settings.

  Examples:

  ```
  https:
    mode: Disabled
  ```

  ```
  https:
    mode: OnlyInURI
  ```

  ```
  https:
    mode: CustomCertificate
    customCertificate:
      secretName: foobar
  ```

  ```
  https:
    mode: CertManager
    certManager:
      clusterIssuerName: letsencrypt
  ```

  - **settings.https.certManager**
    object

    Parameters for certmanager.

    - **settings.https.certManager.clusterIssuerName**
      string

      What ClusterIssuer to use for getting an SSL certificate (currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are supported; you can also define your own).

      Default: `letsencrypt`

      Example:

```
clusterIssuerName: letsencrypt
```

- **settings.https.customCertificate**
  object

  Parameters for custom certificate usage.

  - **settings.https.customCertificate.secretName**
    string

    The name of the secret in the `d8-system` namespace to use with the module's web interface.

    Note that this secret must conform to the kubernetes.io/tls format.

- **settings.https.mode**
  string

  The HTTPS usage mode:

  - `CertManager` — the web UI is accessed over HTTPS using a certificate obtained from a clusterIssuer specified in the `certManager.clusterIssuerName` parameter.
  - `CustomCertificate` — the web UI is accessed over HTTPS using a certificate from the `d8-system` namespace.
  - `Disabled` — in this mode, the module's web interface can only be accessed over HTTP.
  - `OnlyInURI` — the module's web interface will work over HTTP (implies there is an external HTTPS load balancer in front of it that terminates HTTPS traffic). All the links in the `user-authn` will be generated using the HTTPS scheme. The load balancer should provide redirection from HTTP to HTTPS.

  Default: `CertManager`

  Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller of the module web interface.

  An optional parameter; by default, the `modules.ingressClass` global value is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.nodeSelector**
  object

  The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  nodeSelector:
    disktype: ssd
  ```

- **settings.tolerations**
  array of objects

  The same as in the pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  tolerations:
  - key: key1
    operator: Equal
    value: value1
    effect: NoSchedule
  ```

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

The descheduler module

> **i** The module lifecycle stage: General Availability

Every 15 minutes, the module analyzes the cluster state and performs pod eviction according to the conditions described in the active strategies. Evicted pods go through the scheduling process again, considering the current state of the cluster. This helps redistribute workloads according to the chosen strategy.

The module is based on the descheduler project.

Features of the module

- The module can take into account the pod priority class (parameter spec.priorityClassThreshold), restricting its operation to only those pods that have a priority class lower than the specified threshold;
- The module does not evict pods in the following cases:
  - a pod is in the `d8-*` or `kube-system` namespaces;
  - a pod has a `priorityClassName` `system-cluster-critical` or `system-node-critical`;
  - a pod is associated with a local storage;
  - a pod is associated with a DaemonSet;
  - pod eviction will violate Pod Disruption Budget (PDB);
  - there are no available nodes to run the evicted pod.
- Pods with the `Best effort` priority class are evicted before those with `Burstable` and `Guaranteed`.

Descheduler uses parameters with the `labelSelector` syntax from Kubernetes to filter pods and nodes:

- `podLabelSelector` — limits pods by labels;
- `namespaceLabelSelector` — filters pods by namespaces;
- `nodeLabelSelector` — selects nodes by labels.

Metrics provider

If the cluster has a registered `metrics.k8s.io` API group (for example, when metrics-server is installed), the module automatically detects it and enables the **KubernetesMetrics** metrics provider in the `descheduler` policy. This allows to use the **actual** resource consumption data from the Metrics API instead of relying solely on pod requests and limits when the `LowNodeUtilization` strategy is set. Accordingly, the values in thresholds and targetThresholds are then interpreted as a percentage of actual consumption, not of pod requests. The `HighNodeUtilization` strategy is not affected: it always uses pod requests and limits.

No user action is required: the module discovers the `metrics.k8s.io` API group by watching APIService resources and configures the policy accordingly. If metrics-server is installed **after** the module is already running, the `descheduler` Pod will be automatically restarted with the updated policy.

> **i** When `metrics.k8s.io` is not available, the module falls back to the default behavior: resource usage is estimated from pod requests and limits.

Strategies

HighNodeUtilization

> **i** More compactly places pods. Requires scheduler configuration and enabling auto-scaling.
>
> To use `HighNodeUtilization`, you must explicitly specify the high-node-utilization scheduler profile for each pod (this profile cannot be set as the default).

This strategy identifies *under utilized nodes* and evicts pods from them to redistribute them more compactly across fewer nodes.

Under utilized node — A node whose resource usage is below all the threshold values specified in the strategies.highNodeUtilization.thresholds section.

The strategy is enabled by the parameter spec.strategies.highNodeUtilization.enabled.

> **⚠** In GKE, you cannot configure the default scheduler, but you can use the `optimize-utilization` strategy or deploy a second custom scheduler.

⚠ Node resource usage takes into account extended resources and is calculated based on pod requests and limits (requests and limits), not actual consumption. This approach ensures consistency with the kube-scheduler, which uses a similar principle when scheduling pods on nodes.

This strategy always uses pod requests and limits, regardless of whether the `metrics.k8s.io` API is available in the cluster (see Metrics provider).

LowNodeUtilization

i Loads the nodes more evenly.

This strategy identifies *under utilized nodes* and evicts pods from other *over utilized nodes*. The strategy assumes that the evicted pods will be recreated on the under utilized nodes (following normal scheduler behavior).

Under utilized node — A node whose resource usage is below all the threshold values specified in the strategies.lowNodeUtilization.thresholds section.

Over utilized node — A node whose resource usage exceeds at least one of the threshold values specified in the strategies.lowNodeUtilization.targetThresholds section.

Nodes with resource usage in the range between `thresholds` and `targetThresholds` are considered optimally utilized. Pods on these nodes will not be evicted.

The strategy is enabled by the parameter spec.strategies.lowNodeUtilization.enabled.

⚠ If the `metrics.k8s.io` API is **not** available in the cluster, node resource usage takes into account extended resources and is calculated based on pod requests and limits (requests and limits), not actual consumption. This approach ensures consistency with the kube-scheduler, which uses a similar principle when scheduling pods on nodes.

If the `metrics.k8s.io` API **is** available (see Metrics provider), the strategy uses the **actual** resource consumption from the Metrics API *instead of* requests and limits. In this case the values used by the descheduler are close to what `kubectl top` shows, but they no longer match the kube-scheduler's view: a node whose requests are exhausted may still be considered under utilized, and its pods will not be evicted.

RemoveDuplicates

i Prevents multiple pods from the same controller (ReplicaSet, ReplicationController, StatefulSet) or the same Job from running on the same node.

The strategy ensures that no more than one pod of a ReplicaSet, ReplicationController, StatefulSet, or pods of a single Job is running on the same node. If there are two or more such pods, the module evicts the excess pods so that they are better distributed across the cluster.

The situation can occur if some nodes in the cluster have failed for any reason, and the pods from those nodes have been moved to other nodes. Once the failed nodes become available again to accept load, this strategy can be used to evict duplicate pods from other nodes.

The strategy is enabled by the parameter strategies.removeDuplicates.enabled.

RemovePodsHavingTooManyRestarts

The strategy evicts pods having too many restarts from nodes. Pods become eviction candidates when the total number of restarts across all containers, including init containers, exceeds the `podRestartThreshold` threshold. This strategy is useful for evicting pods in the `CrashLoopBackOff` state or with repeated failures, as well as for freeing up resources and allowing fresh pods to be scheduled on potentially healthier nodes.

RemovePodsViolatingInterPodAntiAffinity

i Evicts pods violating inter-pod affinity and anti-affinity rules to ensure compliance.

The strategy ensures that pods violating inter-pod affinity and anti-affinity rules are evicted from nodes.

For example, if there is **podA** on a node and **podB** and **podC** (running on the same node) have anti-affinity rules which prohibit them to run on the same node, then **podA** will be evicted from the node so that **podB** and **podC** could run. This issue could happen, when the anti-affinity rules for **podB** and **podC** are created when they are

already running on node.

The strategy is enabled by the parameter spec.strategies.highNodeUtilization.enabled.

RemovePodsViolatingNodeAffinity

> **i** Evicts pods violating node affinity rules to ensure compliance.

The strategy makes sure all pods violating node affinity are eventually removed from nodes.

Essentially, depending on the settings of the parameter strategies.removePodsViolatingNodeAffinity.nodeAffinityType, the strategy temporarily implement the rule `requiredDuringSchedulingIgnoredDuringExecution` of the pod's node affinity as the rule `requiredDuringSchedulingRequiredDuringExecution`, and the rule `preferredDuringSchedulingIgnoredDuringExecution` as the rule `preferredDuringSchedulingPreferredDuringExecution`.

Example for `nodeAffinityType: requiredDuringSchedulingIgnoredDuringExecution`. There is a pod scheduled to a node which satisfies the node affinity rule `requiredDuringSchedulingIgnoredDuringExecution` at the time of scheduling. If over time this node no longer satisfies the node affinity rule, and there is another node available that satisfies the node affinity rule, the strategy evicts the pod from the node it was originally scheduled to.

Example for `nodeAffinityType: preferredDuringSchedulingIgnoredDuringExecution`. There is a pod scheduled to a node because at the time of scheduling there were no other nodes that satisfied the node affinity rule `preferredDuringSchedulingIgnoredDuringExecution`. If over time an available node that satisfies this rule appears in the cluster, the strategy evicts the pod from the node it was originally scheduled to.

The strategy is enabled by the parameter strategies.removePodsViolatingNodeAffinity.enabled.

RemovePodsViolatingTopologySpreadConstraint

The strategy ensures that pods violating topology spread constraints are evicted from nodes. It evicts the minimum number of pods required to balance topology domains to within each constraint's `maxSkew`. This is useful for rebalancing pods across zones after a zone outage recovery.

Third-party components

List of third-party software used in the `descheduler` module:

- Descheduler 0.35.1

  License: Apache License 2.0

  A tool to auto-fix scheduling issues Kubernetes does not cover itself.

Configuration

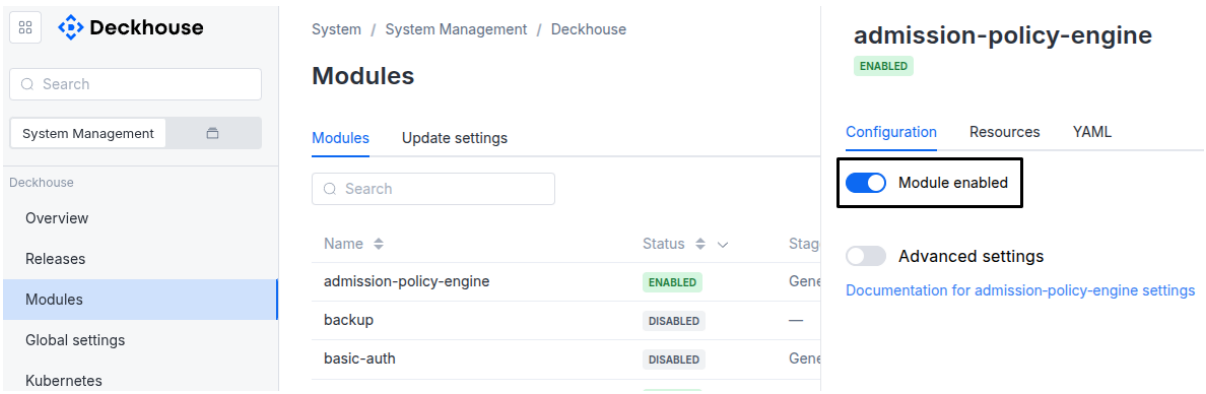
> **i** The module lifecycle stage: General Availability

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `descheduler` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable descheduler
  ```

- **Using ModuleConfig `descheduler`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `descheduler` (create it if necessary);

  Example of a manifest to enable module `descheduler`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: descheduler
  spec:
    enabled: true
  ```

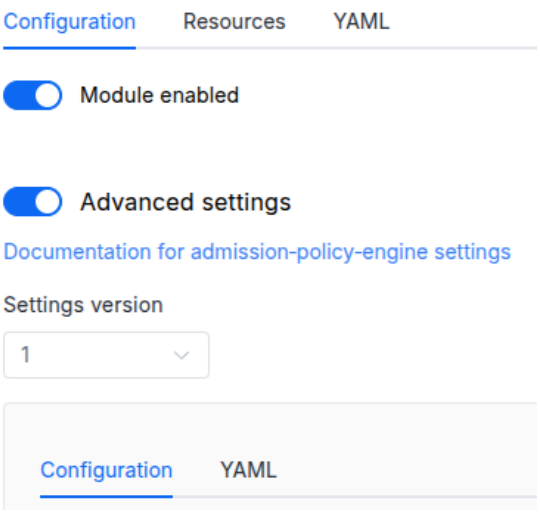
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `descheduler` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `descheduler` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `descheduler`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `descheduler` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc descheduler
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `descheduler` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `descheduler`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: descheduler
spec:
  version: 2
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Delete `settings`.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.deschedulingInterval**
    string

    Interval between pod redistribution runs.

    Supported values:

    - `Frequent`: Runs every 5 minutes. Suitable for clusters where faster pod redistribution is important.
    - `Moderate`: Runs every 15 minutes (default).
    - `Rare`: Runs every 30 minutes. Suitable for clusters where it’s important to minimize the number of pod redistributions.

    Default: `Moderate`

    Allowed values: `Frequent`, `Moderate`, `Rare`

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

Descheduler

Scope: Cluster

- v1alpha2
- v1alpha1

Descheduler is a description of a single descheduler instance.

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
  - **metadata.name**
    string
- **spec**
  object

  Required value

  Defines the behavior of a descheduler instance.

  - **spec.evictLocalStoragePods**
    boolean

    Allows Pods using local storage to be evicted.

    Default: `false`
  - **spec.namespaceLabelSelector**
    object

    Limiting the pods which are processed by namespace labels.

    - **spec.namespaceLabelSelector.matchExpressions**
      array of objects

      List of label expressions that a node should have to qualify for the filter condition.

      Example:

      ```
      matchExpressions:
      - key: tier
        operator: In
        values:
        - production
        - staging
      - key: tier
        operator: NotIn
        values:
        - production
      ```

      - **spec.namespaceLabelSelector.matchExpressions.key**
        string

        A label name.

      - **spec.namespaceLabelSelector.matchExpressions.operator**
        string

        A comparison operator.

        Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

      - **spec.namespaceLabelSelector.matchExpressions.values**
        array of strings

        A label value.

        - **Element of the array**
          string

          Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

          Length: `1..63`
    - **spec.namespaceLabelSelector.matchLabels**
      object
  - **spec.nodeLabelSelector**
    object

    Limiting the pods which are processed to fit evicted pods by labels in set representation. If set, `nodeSelector` must not be set.

    - **spec.nodeLabelSelector.matchExpressions**
      array of objects

List of label expressions that a node should have to qualify for the filter condition.

Example:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
- key: tier
  operator: NotIn
  values:
  - production
```

- **spec.nodeLabelSelector.matchExpressions.key**
  string

  A label name.

- **spec.nodeLabelSelector.matchExpressions.operator**
  string

  A comparison operator.

  Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.nodeLabelSelector.matchExpressions.values**
  array of strings

  A label value.

  - **Element of the array**
    string

    Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

    Length: `1..63`

- **spec.nodeLabelSelector.matchLabels**
  object

- **spec.podLabelSelector**
  object

  Limiting the pods which are processed by labels.

  - **spec.podLabelSelector.matchExpressions**
    array of objects

    List of label expressions that a node should have to qualify for the filter condition.

    Example:

    ```
    matchExpressions:
    - key: tier
      operator: In
      values:
      - production
      - staging
    - key: tier
      operator: NotIn
      values:
      - production
    ```

    - **spec.podLabelSelector.matchExpressions.key**
      string

      A label name.

    - **spec.podLabelSelector.matchExpressions.operator**
      string

      A comparison operator.

      Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

    - **spec.podLabelSelector.matchExpressions.values**
      array of strings

      A label value.

      - **Element of the array**
        string

        Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

        Length: `1..63`

  - **spec.podLabelSelector.matchLabels**
    object

- **spec.priorityClassThreshold**
  object

  Limiting the pods which are processed by priority class. Only pods under the threshold can be evicted.

  You can specify either the name of the priority class (priorityClassThreshold.name), or the actual value of the priority class (priorityThreshold.value).

By default, this threshold is set to the value of `system-cluster-critical` priority class.

- **spec.priorityClassThreshold.name**
  string

  Name of the priority class.

- **spec.priorityClassThreshold.value**
  integer

  Value of the priority class.

- **spec.strategies**
  object

  Required value

  Settings of strategies for the Descheduler instances.

  - **spec.strategies.highNodeUtilization**
    object

    This strategy finds nodes that are under utilized and evicts Pods from the nodes in the hope that these pods will be scheduled compactly into fewer nodes. When combined with node auto-scaling, it helps reduce the number of underutilized nodes. The strategy works with the `MostAllocated` scheduler.

    > **i** In GKE, you cannot configure the default scheduler, but you can use the `optimize-utilization` strategy or deploy a second custom scheduler.

    Node resource usage takes into account extended resources and is based on pod requests and limits, not actual consumption.

    - **spec.strategies.highNodeUtilization.enabled**
      boolean

      Makes the strategy active.

      Default: `false`

    - **spec.strategies.highNodeUtilization.thresholds**
      object

      Sets threshold values to identify to identify under utilized nodes.

      If the resource usage of the node is below **all** threshold values, then the node is considered under utilized.

      If a resource threshold is not specified, it is treated as 100% and does not participate in node utilization calculations.

      - **spec.strategies.highNodeUtilization.thresholds.cpu**
        integer

        CPU fraction in percents

        Allowed values: `0 <= X <= 100`

      - **spec.strategies.highNodeUtilization.thresholds.memory**
        integer

        Memory fraction in percents

        Allowed values: `0 <= X <= 100`

      - **spec.strategies.highNodeUtilization.thresholds.pods**
        integer

        Pods count in percents

        Allowed values: `0 <= X <= 100`

  - **spec.strategies.lowNodeUtilization**
    object

    This strategy identifies *under utilized nodes* and evicts pods from other *over utilized nodes*. The strategy assumes that the evicted pods will be recreated on the under utilized nodes (following normal scheduler behavior).

    Under utilized node — A node whose resource usage is below all the threshold values specified in the thresholds section.

    Over utilized node — A node whose resource usage exceeds at least one of the threshold values specified in the targetThresholds section.

    Node resource usage takes into account extended resources and is based on pod requests and limits, not actual consumption.

    If the `metrics.k8s.io` API group is registered in the cluster, the module automatically switches this strategy to the **actual** resource consumption from the Metrics API.

    - **spec.strategies.lowNodeUtilization.enabled**
      boolean

      Makes the strategy active.

      Default: `false`

    - **spec.strategies.lowNodeUtilization.targetThresholds**
      object

      Sets threshold values to identify to identify over utilized nodes.

      If the resource usage of the node exceeds at least one of the threshold values, then the node is considered over utilized.

If a resource threshold is not specified, it is treated as 100% and does not participate in node utilization calculations.

- **spec.strategies.lowNodeUtilization.targetThresholds.cpu**
  integer

  CPU fraction in percents

  Allowed values: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization.targetThresholds.memory**
  integer

  Memory fraction in percents

  Allowed values: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization.targetThresholds.pods**
  integer

  Pods count in percents

  Allowed values: `0 <= X <= 100`

- **spec.strategies.lowNodeUtilization.thresholds**
  object

  Sets threshold values to identify to identify under utilized nodes.

  If the resource usage of the node is below **all** threshold values, then the node is considered under utilized.

  If a resource threshold is not specified, it is treated as 100% and does not participate in node utilization calculations.

  - **spec.strategies.lowNodeUtilization.thresholds.cpu**
    integer

    CPU fraction in percents

    Allowed values: `0 <= X <= 100`

  - **spec.strategies.lowNodeUtilization.thresholds.memory**
    integer

    Memory fraction in percents

    Allowed values: `0 <= X <= 100`

  - **spec.strategies.lowNodeUtilization.thresholds.pods**
    integer

    Pods count in percents

    Allowed values: `0 <= X <= 100`

- **spec.strategies.removeDuplicates**
  object

  The strategy ensures that no more than one pod of a ReplicaSet, ReplicationController, StatefulSet, or pods of a single Job is running on the same node. If there are two or more such pods, the module evicts the excess pods so that they are better distributed across the cluster.

  - **spec.strategies.removeDuplicates.enabled**
    boolean

    Makes the strategy active.

    Default: `false`

- **spec.strategies.removePodsHavingTooManyRestarts**
  object

  The strategy evicts pods having too many restarts from nodes.

  Pods become eviction candidates when the total number of restarts across all containers, including init containers, exceeds the `podRestartThreshold` threshold.

  This strategy is useful for evicting pods in the `CrashLoopBackOff` state or with repeated failures, as well as for freeing up resources and allowing fresh pods to be scheduled on potentially healthier nodes.

  - **spec.strategies.removePodsHavingTooManyRestarts.enabled**
    boolean

    Makes the strategy active.

    Default: `false`

  - **spec.strategies.removePodsHavingTooManyRestarts.podRestartThreshold**
    integer

    Threshold for the total number of restarts across all pod containers, including init containers.

    If the number of restarts exceeds this value, the pod is considered unhealthy and becomes a candidate for eviction.

    Default: `100`

    Allowed values: `1 <= X`

- **spec.strategies.removePodsViolatingInterPodAntiAffinity**
  object

  The strategy ensures that pods violating inter-pod affinity and anti-affinity rules are evicted from nodes.

  - **spec.strategies.removePodsViolatingInterPodAntiAffinity.enabled**
    boolean

    Makes the strategy active.

Default: `false`

- **spec.strategies.removePodsViolatingNodeAffinity**
  object

  The strategy makes sure all pods violating node affinity are eventually removed from nodes.

  Essentially, depending on the settings of the parameter nodeAffinityType, the strategy temporarily implement the rule `requiredDuringSchedulingIgnoredDuringExecution` of the pod's node affinity as the rule `requiredDuringSchedulingRequiredDuringExecution`, and the rule `preferredDuringSchedulingIgnoredDuringExecution` as the rule `preferredDuringSchedulingPreferredDuringExecution`.

  - **spec.strategies.removePodsViolatingNodeAffinity.enabled**
    boolean

    Makes the strategy active.

    Default: `false`

  - **spec.strategies.removePodsViolatingNodeAffinity.nodeAffinityType**
    array of strings

    Defines the list of node affinity rules used.

    Default: `["requiredDuringSchedulingIgnoredDuringExecution"]`

    - **Element of the array**
      string

      Allowed values: `requiredDuringSchedulingIgnoredDuringExecution`, `preferredDuringSchedulingIgnoredDuringExecution`

- **spec.strategies.removePodsViolatingTopologySpreadConstraint**
  object

  The strategy ensures that pods violating topology spread constraints are evicted from nodes. It evicts the minimum number of pods required to balance topology domains to within each constraint's `maxSkew`.

  This is useful for rebalancing pods across zones after a zone outage recovery.

  - **spec.strategies.removePodsViolatingTopologySpreadConstraint.constraints**
    array of strings

    List of topology spread constraint types to consider when deciding which pods to evict.

    Topology spread constraints control how pods are distributed across your cluster topology (e.g. regions, zones, nodes).

    By default, only `DoNotSchedule` (hard constraints) are considered — these are constraints that the scheduler strictly enforces: a pod will NOT be placed if doing so would violate the constraint. `Descheduler` will evict pods that are currently violating such constraints (e.g. due to node failures or cluster topology changes after initial scheduling).

    Optionally, `ScheduleAnyway` (soft constraints) can also be included — these are constraints that the scheduler tries to satisfy but does not strictly enforce: a pod WILL be placed even if the constraint cannot be satisfied, though the scheduler will attempt to minimize the imbalance. Including this type causes `descheduler` to also evict pods that violate soft constraints.

    Default: `["DoNotSchedule"]`

    - **Element of the array**
      string

      Allowed values: `DoNotSchedule`, `ScheduleAnyway`

  - **spec.strategies.removePodsViolatingTopologySpreadConstraint.enabled**
    boolean

    Makes the strategy active.

    Default: `false`

  - **spec.strategies.removePodsViolatingTopologySpreadConstraint.topologyBalanceNodeFit**
    boolean

    If set to `true`, the strategy will consider whether the pods can fit on other nodes when balancing topology domains.

    Default: `true`

Deprecated resource. Support for the resource might be removed in a later release.

Descheduler is a description of a single descheduler instance.

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
  - **metadata.name**
    string
- **spec**
  object

  Required value

  Defines the behavior of a descheduler instance.

  - **spec.deploymentTemplate**
    object

    Defines Template of a descheduler Deployment.

- **spec.deploymentTemplate.nodeSelector**
  object
- **spec.deploymentTemplate.tolerations**
  array of objects
  - **spec.deploymentTemplate.tolerations.effect**
    string
  - **spec.deploymentTemplate.tolerations.key**
    string
  - **spec.deploymentTemplate.tolerations.operator**
    string
  - **spec.deploymentTemplate.tolerations.tolerationSeconds**
    integer
  - **spec.deploymentTemplate.tolerations.value**
    string

- **spec.deschedulerPolicy**
  object

  `globalParameters` and `strategies` follow the descheduler's documentation.

  - **spec.deschedulerPolicy.globalParameters**
    object

    Parameters that apply to all policies.

    - **spec.deschedulerPolicy.globalParameters.evictFailedBarePods**
      boolean

      Allows Pods without `ownerReferences` and in `failed` phase to be evicted.

    - **spec.deschedulerPolicy.globalParameters.evictLocalStoragePods**
      boolean

      Allows Pods using local storage to be evicted.

    - **spec.deschedulerPolicy.globalParameters.evictSystemCriticalPods**
      boolean

      Allows eviction of Pods of any priority (including Kubernetes system Pods).

    - **spec.deschedulerPolicy.globalParameters.ignorePvcPods**
      boolean

      Prevents Pods with PVCs from being evicted.

    - **spec.deschedulerPolicy.globalParameters.maxNoOfPodsToEvictPerNamespace**
      integer

      Restricts maximum of Pods to be evicted per namespace.

    - **spec.deschedulerPolicy.globalParameters.maxNoOfPodsToEvictPerNode**
      integer

      Restricts maximum of Pods to be evicted per node.

    - **spec.deschedulerPolicy.globalParameters.nodeSelector**
      string
  - **spec.deschedulerPolicy.strategies**
    object

    List of strategies with corresponding parameters for a given Descheduler instances.

    - **spec.deschedulerPolicy.strategies.highNodeUtilization**
      object

      This strategy finds nodes that are under utilized and evicts Pods from the nodes in the hope that these Pods will be scheduled compactly into fewer nodes.

      - **spec.deschedulerPolicy.strategies.highNodeUtilization.enabled**
        boolean

        Required value

      - **spec.deschedulerPolicy.strategies.highNodeUtilization.namespaceFilter**
        object

        Restricts Namespaces to which this strategy applies.

      - **spec.deschedulerPolicy.strategies.highNodeUtilization.nodeFilter**
        object

        Filters Nodes to which the strategy applies.

      - **spec.deschedulerPolicy.strategies.highNodeUtilization.nodeFit**
        boolean

        If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

      - **spec.deschedulerPolicy.strategies.highNodeUtilization.priorityFilter**
        object

        Only Pods with priority lower than this will be descheduled.

    - **spec.deschedulerPolicy.strategies.lowNodeUtilization**
      object

      This strategy finds nodes that are under utilized and evicts Pods, if possible, from other nodes in the hope that recreation of evicted Pods will be scheduled on these underutilized nodes.

      - **spec.deschedulerPolicy.strategies.lowNodeUtilization.enabled**
        boolean

        Required value

- **spec.deschedulerPolicy.strategies.lowNodeUtilization.namespaceFilter**
  object

  Restricts Namespaces to which this strategy applies.

- **spec.deschedulerPolicy.strategies.lowNodeUtilization.nodeFilter**
  object

  Filters Nodes to which the strategy applies.

- **spec.deschedulerPolicy.strategies.lowNodeUtilization.nodeFit**
  boolean

  If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

- **spec.deschedulerPolicy.strategies.lowNodeUtilization.priorityFilter**
  object

  Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removeDuplicates**
  object

  This strategy makes sure that there is only one Pod associated with a ReplicaSet (RS), ReplicationController (RC), StatefulSet, or Job running on the same node.

  - **spec.deschedulerPolicy.strategies.removeDuplicates.enabled**
    boolean

    Required value

  - **spec.deschedulerPolicy.strategies.removeDuplicates.namespaceFilter**
    object

    Restricts Namespaces to which this strategy applies.

  - **spec.deschedulerPolicy.strategies.removeDuplicates.nodeFilter**
    object

    Filters Nodes to which the strategy applies.

  - **spec.deschedulerPolicy.strategies.removeDuplicates.nodeFit**
    boolean

    If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

  - **spec.deschedulerPolicy.strategies.removeDuplicates.priorityFilter**
    object

    Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removeFailedPods**
  object

  This strategy evicts Pods that are in `failed` status phase.

  - **spec.deschedulerPolicy.strategies.removeFailedPods.enabled**
    boolean

    Required value

  - **spec.deschedulerPolicy.strategies.removeFailedPods.namespaceFilter**
    object

    Restricts Namespaces to which this strategy applies.

  - **spec.deschedulerPolicy.strategies.removeFailedPods.nodeFilter**
    object

    Filters Nodes to which the strategy applies.

  - **spec.deschedulerPolicy.strategies.removeFailedPods.nodeFit**
    boolean

    If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

  - **spec.deschedulerPolicy.strategies.removeFailedPods.priorityFilter**
    object

    Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts**
  object

  This strategy makes sure that Pods having too many restarts are removed from nodes.

  - **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.enabled**
    boolean

    Required value

  - **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.namespaceFilter**
    object

    Restricts Namespaces to which this strategy applies.

  - **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.nodeFilter**
    object

    Filters Nodes to which the strategy applies.

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.nodeFit**
  boolean

  If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

- **spec.deschedulerPolicy.strategies.removePodsHavingTooManyRestarts.priorityFilter**
  object

  Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity**
  object

  This strategy makes sure that Pods violating interpod anti-affinity are removed from nodes.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.enabled**
    boolean

    Required value

  - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.namespaceFilter**
    object

    Restricts Namespaces to which this strategy applies.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.nodeFilter**
    object

    Filters Nodes to which the strategy applies.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.nodeFit**
    boolean

    If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingInterPodAntiAffinity.priorityFilter**
    object

    Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity**
  object

  This strategy makes sure all Pods violating node affinity are eventually removed from nodes.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.enabled**
    boolean

    Required value

  - **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.namespaceFilter**
    object

    Restricts Namespaces to which this strategy applies.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.nodeFilter**
    object

    Filters Nodes to which the strategy applies.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.nodeFit**
    boolean

    If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingNodeAffinity.priorityFilter**
    object

    Only Pods with priority lower than this will be descheduled.

- **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints**
  object

  This strategy makes sure that Pods violating `NoSchedule` taints on nodes are removed.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.enabled**
    boolean

    Required value

  - **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.namespaceFilter**
    object

    Restricts Namespaces to which this strategy applies.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.nodeFilter**
    object

    Filters Nodes to which the strategy applies.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.nodeFit**
    boolean

    If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.

  - **spec.deschedulerPolicy.strategies.removePodsViolatingNodeTaints.priorityFilter**
    object

    Only Pods with priority lower than this will be descheduled.

        - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint**
          object

          This strategy makes sure that Pods violating topology spread constraints are evicted from nodes.

          - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.enabled**
            boolean

            Required value
          - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.namespaceFilter**
            object

            Restricts Namespaces to which this strategy applies.
          - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.nodeFilter**
            object

            Filters Nodes to which the strategy applies.
          - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.nodeFit**
            boolean

            If set to `true`, the descheduler will consider whether or not the Pods that meet eviction criteria will fit on other nodes before evicting them.
          - **spec.deschedulerPolicy.strategies.removePodsViolatingTopologySpreadConstraint.priorityFilter**
            object

            Only Pods with priority lower than this will be descheduled.

- **status**
  object

  Most recently observed status of a descheduler instance.

  - **status.ready**
    boolean

    Required value

Examples

> **i** The module lifecycle stage: General Availability

Configuring the pod redistribution interval

To set how often the `descheduler` module runs the pod redistribution cycle, use the `deschedulingInterval` parameter.

For example, to run `descheduler` every 5 minutes, set `deschedulingInterval: Frequent`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: descheduler
spec:
  enabled: true
  settings:
    deschedulingInterval: Frequent
```

Supported values:

- `Frequent`: Runs every 5 minutes. Suitable for clusters where faster pod redistribution is important.
- `Moderate`: Runs every 15 minutes (default).
- `Rare`: Runs every 30 minutes. Suitable for clusters where it's important to minimize the number of pod redistributions.

Example LowNodeUtilization strategy

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: low-node-utilization
spec:
  strategies:
    lowNodeUtilization:
      enabled: true
      thresholds:
        cpu: 20
      targetThresholds:
        cpu: 50
```

Example HighNodeUtilization strategy

```
---
apiVersion: deckhouse.io/v1alpha2
kind: Descheduler
metadata:
  name: high-node-utilization
spec:
  strategies:
    highNodeUtilization:
      enabled: true
      thresholds:
        cpu: 50
        memory: 50
```

The documentation module

> **i** The module lifecycle stage: General Availability

> **i** Web interfaces associated with the module: documentation

The `documentation` module creates a documentation web UI for the Deckhouse version currently used in a cluster.

This can be useful when Deckhouse works in a network with limited Internet access.

To get the web interface address, in the publicDomainTemplate template of the Deckhouse global configuration parameter, replace the `%s` key with `documentation`.

For example, if `publicDomainTemplate` is set as `%s-kube.company.my`, then the documentation web interface will be available at `documentation-kube.company.my`.

Configuration

> **i** The module lifecycle stage: General Availability

The module does not have any mandatory parameters.

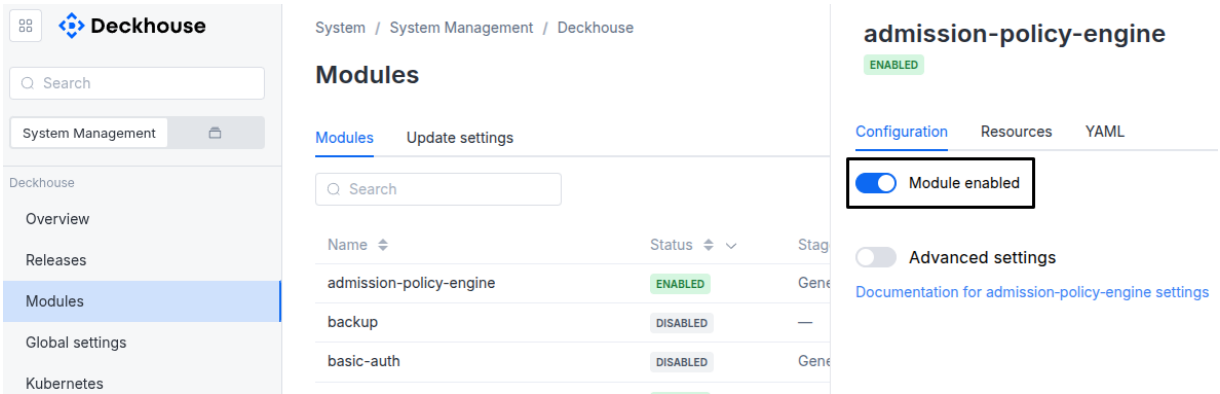
The module has 2 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `documentation` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable documentation
  ```

- **Using ModuleConfig `documentation`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `documentation` (create it if necessary);

  Example of a manifest to enable module `documentation`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: documentation
spec:
  enabled: true
```

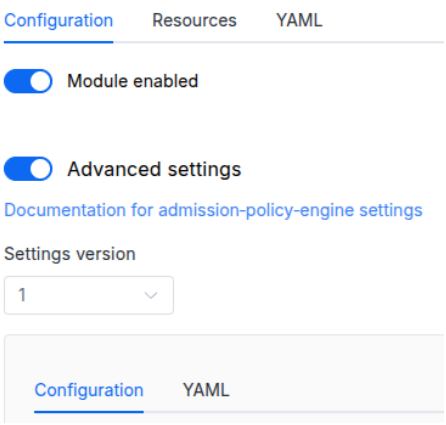
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `documentation` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `documentation` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `documentation`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `documentation` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc documentation
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `documentation` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `documentation`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: documentation
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.auth**
    object

    Parameters to authenticate and authorize access to the documentation web interface.

    - **settings.auth.allowedUserEmails**
      array of strings

      An array of emails of users that can access module’s public web interfaces.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

    - **settings.auth.allowedUserGroups**
      array of strings

An array of groups whose users can browse the documentation.

This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

> **Caution!** Note that you must add those groups to the appropriate field in the `DexProvider` config if this module is used together with the `user-authn` one.

Default: `[]`

Examples:

```
allowedUserGroups:
- admin
- users
```

```
allowedUserGroups: []
```

- **settings.auth.externalAuthentication**
  object

  Parameters to enable external authentication based on the Ingress NGINX external-auth mechanism that uses the Nginx auth_request module.

  > External authentication is enabled automatically if the user-authn module is enabled.

  - **settings.auth.externalAuthentication.authSignInURL**
    string

    The URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response.

    Default:

    Example:

    ```
    authSignInURL: https://$host/dex-authenticator/sign_in
    ```

  - **settings.auth.externalAuthentication.authURL**
    string

    The URL of the authentication service.

    If the user is authenticated, the service should return an HTTP 200 response code.

    Default:

    Example:

    ```
    authURL: https://documentation-dex-authenticator.d8-system.svc.cluster.local/dex-authenticator/auth
    ```

- **settings.https**
  object

  What certificate type to use.

  This parameter completely overrides the `global.modules.https` settings.

  Examples:

  ```
  https:
    mode: Disabled
  ```

  ```
  https:
    mode: OnlyInURI
  ```

  ```
  https:
    mode: CustomCertificate
    customCertificate:
      secretName: foobar
  ```

  ```
  https:
    mode: CertManager
    certManager:
      clusterIssuerName: letsencrypt
  ```

  - **settings.https.certManager**
    object

    Parameters for certmanager.

    - **settings.https.certManager.clusterIssuerName**
      string

      What ClusterIssuer to use for getting an SSL certificate (currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available; also, you can define your own).

      Default: `letsencrypt`

Example:

```
clusterIssuerName: letsencrypt
```

- **settings.https.customCertificate**
  object

  Parameters for custom certificate usage.

  - **settings.https.customCertificate.secretName**
    string

    The name of the secret in the `d8-system` namespace to use with the documentation web UI.

    This secret must have the kubernetes.io/tls format.

- **settings.https.mode**
  string

  The HTTPS usage mode:

  - `CertManager` — the web UI is accessed over HTTPS using a certificate obtained from a clusterIssuer specified in the `certManager.clusterIssuerName` parameter.
  - `CustomCertificate` — the web UI is accessed over HTTPS using a certificate from the `d8-system` namespace.
  - `Disabled` — in this mode, the documentation web UI can only be accessed over HTTP.
  - `OnlyInURI` — HTTP access with an external HTTPS balancer. The balancer terminates HTTPS, and all links in `user-authn` are generated with the HTTPS scheme. The balancer must provide redirection from HTTP to HTTPS.

  Default: `CertManager`

  Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller of the documentation web UI.

  An optional parameter; by default, the `modules.ingressClass` global value is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.nodeSelector**
  object

  The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  nodeSelector:
    disktype: ssd
  ```

- **settings.tolerations**
  array of objects

  The same as in the pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  tolerations:
  - key: key1
    operator: Equal
    value: value1
    effect: NoSchedule
  ```

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

Authentication

user-authn module provides authentication by default. Also, externalAuthentication can be configured (see below). If these options are disabled, the module will use basic auth with the auto-generated password.

Use d8 k to see password:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values documentation -o json | jq
'.internal.auth.password'
```

Delete the Secret to re-generate password:

```
d8 k -n d8-system delete secret/documentation-basic-auth
```

> **Note!** The `auth.password` parameter is deprecated.

Examples

> The module lifecycle stage: General Availability

An example of the module configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: documentation
spec:
  version: 1
  enabled: true
  settings:
    nodeSelector:
      node-role/system: ""
    tolerations:
    - key: dedicated.deckhouse.io
      operator: Equal
      value: system
    externalAuthentication:
      authURL: "https://<applicationDomain>/auth"
      authSignInURL: "https://<applicationDomain>/sign-in"
      authResponseHeaders: "Authorization"
```

The extended-monitoring module

> The module lifecycle stage: General Availability
> The module has requirements for installation

The `extended-monitoring` module extends cluster monitoring capabilities with additional Prometheus exporters, which allow you to identify potential problems before they affect the operation of services.

Module features:

- Advanced Metrics Collection — collects additional metrics, and also includes ready-made alerts and dashboards that allow you to detect and diagnose incidents faster:
  - collects and expounds metrics for free space and inodes on nodes, as well as for objects with a label `extended-monitoring.deckhouse.io/enabled =""` in the namespace;
  - automatically generates alerts when the thresholds are reached.
- Container image monitoring:
  - adds metrics and sends alerts about unavailability of container images to registry for all types of workload (`Deployments`, `StatefulSets`, `DaemonSets`, `CronJobs`);
  - helps to find out in advance about possible problems with launching or updating pods.
- Cluster Events — collects Kubernetes events and displays them as metrics, which allows you to track the dynamics of changes and respond faster to incidents:
- Certificate control:
  - scans the cluster's Secrets and generates metrics about the expiration of x509 certificates;
  - allows you not to miss critical moments and update certificates on time, avoiding application downtime due to expired certificates.

Configuration

> The module lifecycle stage: General Availability
> The module has requirements for installation

How to use `extended-monitoring-exporter`

Attach the `extended-monitoring.deckhouse.io/enabled` label to the Namespace to enable the export of extended monitoring metrics. You can do it by:

- adding the appropriate helm-chart to the project (recommended method);
- adding it to `.gitlab-ci.yml` (kubectl patch/create);
- attaching it manually (`d8 k label namespace my-app-production extended-monitoring.deckhouse.io/enabled=""`).
- configuring via namespace-configurator module.

Any of the methods above would result in the emergence of the default metrics (+ any custom metrics with the `threshold.extended-monitoring.deckhouse.io/` prefix) for all supported Kubernetes objects in the target namespace. Note that monitoring is enabled automatically for a number of non-namespaced Kubernetes objects described below.

You can also add custom labels with the specified value to `threshold.extended-monitoring.deckhouse.io/something` Kubernetes objects, e.g., `d8 k label pod test threshold.extended-monitoring.deckhouse.io/disk-inodes-warning=30`. In this case, the label value will replace the default one.

If you want to override the default thresholds for all objects in a namespace, you can set the `threshold.extended-monitoring.deckhouse.io/` label at the namespace level. For example: `d8 k label namespace my-app-production threshold.extended-monitoring.deckhouse.io/5xx-warning=20` This will replace the default value for all objects in the namespace that do not already have this label set.

You can disable monitoring on a per-object basis by adding the `extended-monitoring.deckhouse.io/enabled=false` label to it. Thus,

the default labels will also be disabled (as well as label-based alerts).

Standard labels and supported Kubernetes objects

Below is the list of labels used in Prometheus Rules and their default values.

Note, that all the labels start with the `threshold.extended-monitoring.deckhouse.io/` prefix. The value specified in a label is a number that sets the alert trigger threshold.

For example, the label `threshold.extended-monitoring.deckhouse.io/5xx-warning: "5"` on the Ingress resource changes the alert threshold from 10% (default) to 5%.

Non-namespaced Kubernetes objects

Non-namespaced Kubernetes objects do not need labels on the namespace, and monitoring on them is enabled by default when the module is enabled.

Node

Label	Type	Default value
disk-bytes-warning	int (percent)	70
disk-bytes-critical	int (percent)	80
disk-inodes-warning	int (percent)	90
disk-inodes-critical	int (percent)	95
load-average-per-core-warning	int	3
load-average-per-core-critical	int	10

Caution! These labels **do not** apply to `imagefs` (`/var/lib/docker` by default) and `nodefs` (`/var/lib/kubelet` by default) volumes. The thresholds for these volumes are configured completely automatically according to the kubelet's eviction thresholds. The default values are available here; for more info, see the exporter.

Namespaced Kubernetes objects

Pod

Label	Type	Default value
disk-bytes-warning	int (percent)	85
disk-bytes-critical	int (percent)	95
disk-inodes-warning	int (percent)	85
disk-inodes-critical	int (percent)	90

Ingress

Label	Type	Default value
5xx-warning	int (percent)	10
5xx-critical	int (percent)	20

Deployment

Label	Type	Default value
replicas-not-ready	int (count)	0

The threshold implies the number of unavailable replicas **in addition** to maxUnavailable. This threshold will be triggered if the number of unavailable replicas is greater than `maxUnavailable` by the amount specified. Suppose `replicas-not-ready` is 0. In this case, the threshold will be triggered if the number of unavailable replicas is greater than `maxUnavailable`. If `replicas-not-ready` is set to 1, then the threshold will be triggered if the number of unavailable replicas is greater than `maxUnavailable` + 1. This way, you can fine-tune this parameter for specific Deployments (that may be unavailable) in the namespace with the extended monitoring enabled to avoid getting excessive alerts.

StatefulSet

Label	Type	Default value
replicas-not-ready	int (count)	0

The threshold implies the number of unavailable replicas **in addition** to maxUnavailable (see the comments on Deployment).

DaemonSet

Label	Type	Default value
replicas-not-ready	int (count)	0

The threshold implies the number of unavailable replicas **in addition** to maxUnavailable (see the comments on Deployment).

CronJob

Note that only the deactivation using the `extended-monitoring.deckhouse.io/enabled=false` label is supported.

How does it work?

The module exports specific Kubernetes object labels to Prometheus. It allows you to improve Prometheus rules by adding the thresholds for triggering alerts. Using metrics that this module exports, you can, e.g., replace the “magic” constants in rules.

Before:

```
(
  kube_statefulset_status_replicas - kube_statefulset_status_replicas_ready
)
> 1
```

After:

```
(
  kube_statefulset_status_replicas - kube_statefulset_status_replicas_ready
)
> on (namespace, statefulset)
(
  max by (namespace, statefulset) (extended_monitoring_statefulset_threshold{threshold="replicas-not-ready"})
)
```

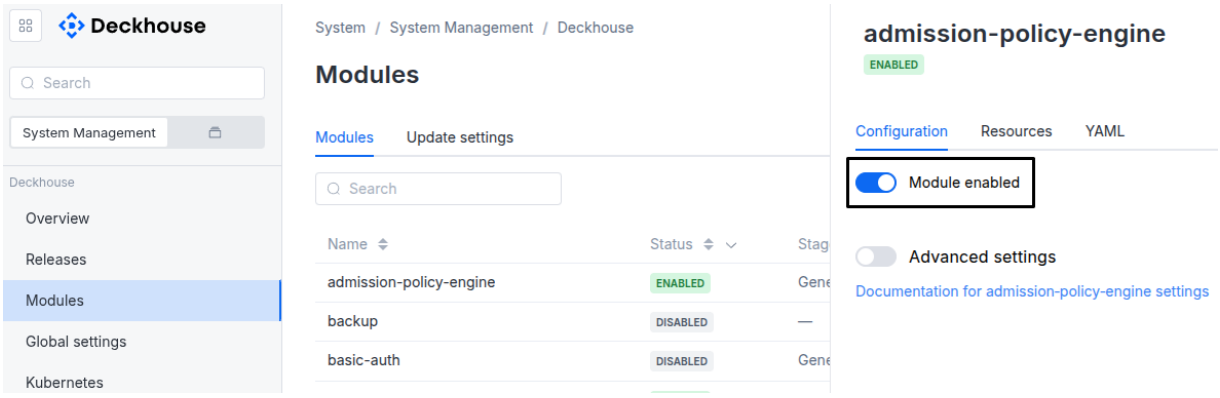
The module has 58 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `extended-monitoring` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable extended-monitoring
  ```

- **Using ModuleConfig `extended-monitoring`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `extended-monitoring` (create it if necessary);

  Example of a manifest to enable module `extended-monitoring`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: extended-monitoring
  spec:
    enabled: true
  ```

How to configure the module...

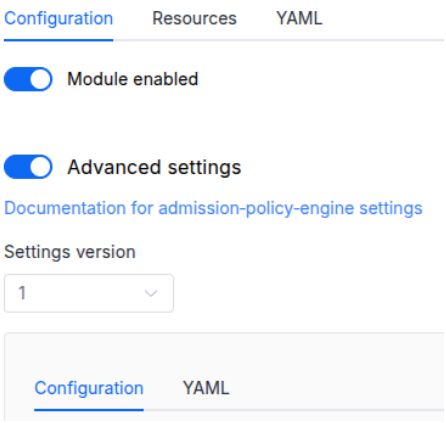
You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `extended-monitoring` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML

format on the “YAML” tab, **excluding the settings section**. Save the changes.

Example:



You can also edit the ModuleConfig object `extended-monitoring` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `extended-monitoring`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `extended-monitoring` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc extended-monitoring
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `extended-monitoring` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `extended-monitoring`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: extended-monitoring
  spec:
    version: 2
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the versions of other modules:

- `prometheus`: any version.

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  If the `.imageAvailability.skipRegistryCertVerification` field is set to `true`, add `.imageAvailability.registry.tlsConfig.insecureSkipVerify=true`. Then, delete the `skipRegistryCertVerification` field from the object. If the `.imageAvailability` object becomes empty after this change, delete it.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.certificates**
    object

    Settings for monitoring the certificates in the Kubernetes cluster.

    - **settings.certificates.exporterEnabled**
      boolean

      Enables x509-certificate-exporter.

      Default: `false`

- **settings.events**
  object

  Settings for monitoring the events in the Kubernetes cluster.

  - **settings.events.exporterEnabled**
    boolean

    Enables eventsExporter.

    Default: `false`

  - **settings.events.severityLevel**
    string

    Whether to expose only crucial events.

    Default: `OnlyWarnings`

    Allowed values: `All`, `OnlyWarnings`

- **settings.imageAvailability**
  object

  Settings for monitoring the availability of images in the cluster.

  - **settings.imageAvailability.defaultRegistry**
    string

    Default registry to use when an image name is not fully qualified.

    If not set, index.docker.io is used by default.

    Default: `index.docker.io`

  - **settings.imageAvailability.exporterEnabled**
    boolean

    Enables imageAvailabilityExporter.

    Default: `true`

  - **settings.imageAvailability.forceCheckDisabledControllers**
    array of strings

    A list of controller kinds for which image is forcibly checked, even when workloads are disabled or suspended.

    Specify `All` to check all controller kinds.

    Example:

    ```
    forceCheckDisabledControllers:
    - Deployment
    - StatefulSet
    ```

    - **Element of the array**
      string

      Allowed values: `Deployment`, `StatefulSet`, `DaemonSet`, `CronJob`, `All`

  - **settings.imageAvailability.ignoredImages**
    array of strings

    A list of images to ignore when checking the presence in the registry, e.g., `alpine:3.12` or `quay.io/test/test:v1.1`.

    Example:

    ```
    ignoredImages:
    - alpine:3.10
    - alpine:3.2
    ```

  - **settings.imageAvailability.imageCheckInterval**
    string

    Interval for checking container registries for image availability.

    Supports Go-style duration format (e.g. “10s”, “10m”, “10h”).

    If not specified, the default value is “60s”.

    Default: `60s`

  - **settings.imageAvailability.mirrors**
    array of objects

    List of mirrors for container registries.

    Example:

    ```
    mirrors:
    - original: docker.io
      mirror: mirror.gcr.io
    - original: internal-registry.com
      mirror: mirror.internal-registry.com
    ```

    - **settings.imageAvailability.mirrors.mirror**
      string

      Required value

        - **settings.imageAvailability.mirrors.original**
          string

          Required value

    - **settings.imageAvailability.registry**
      object

      Connection settings for container registry.

      - **settings.imageAvailability.registry.scheme**
        string

        Container registry access scheme.

        Default: `HTTPS`

        Allowed values: `HTTP`, `HTTPS`

      - **settings.imageAvailability.registry.tlsConfig**
        object

        Connection settings for container registry.

        - **settings.imageAvailability.registry.tlsConfig.ca**
          string

          Root CA certificate to validate the container registry's HTTPS certificate (if self-signed certificates are used).

        - **settings.imageAvailability.registry.tlsConfig.insecureSkipVerify**
          boolean

          Whether to skip the verification of the container registry certificate.

          Default: `false`

  - **settings.nodeSelector**
    object

    The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

    If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations**
    array of objects

    The same as in the pods' `spec.tolerations` parameter in Kubernetes;

    If the parameter is omitted or `false`, it will be determined automatically.

    - **settings.tolerations.effect**
      string
    - **settings.tolerations.key**
      string
    - **settings.tolerations.operator**
      string
    - **settings.tolerations.tolerationSeconds**
      integer
    - **settings.tolerations.value**
      string

The extended monitoring module: FAQ

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

How to switch to HTTP instead of HTTPS checks of my registry?

To change the protocol used for checking your container registry from HTTPS to HTTP, change the `settings.imageAvailability.registry.scheme` parameter in the module configuration.

For detailed instructions, please refer to the module configuration documentation.

The ingress-nginx module

> **i** The module lifecycle stage: General Availability

Installs and manages the Ingress NGINX Controller using custom resources. The module installs the Ingress NGINX Controller in the HA mode if there is more than one node. In doing so, it takes into account all the aspects of cloud and bare-metal infrastructure and various types of Kubernetes clusters.

The module supports running and configuring several Ingress NGINX Controllers simultaneously (one of the controllers is the **primary** one; you can create any number of **additional** controllers). This approach allows you to separate extranet and intranet Ingress resources of applications.

> **i** In 2025, Ingress NGINX was placed in maintenance mode, with no plans for active development of new features. Further evolution of inbound traffic load balancing in Kubernetes is focused on the Gateway API.
>
> This does not apply to the module as part of Deckhouse Kubernetes Platform: the module is maintained by the Deckhouse team, including security updates. For details, see Module support and security.

Current nginx version is 1.30.3.

Traffic routing

Traffic to `ingress-nginx` can be routed in several ways:

- Directly without the use of an external load balancer
- Using an external LoadBalancer; the following LB varieties are supported:
  - Qrator
  - Cloudflare
  - AWS LB
  - GCE LB
  - ACS LB
  - Yandex LB
  - OpenStack LB

Supported Protocols

The module allows the following protocols to be used for receiving and further routing traffic:

- HTTP/HTTPS: The primary traffic routing protocol. You can select between HTTP/1.1, HTTP/2, and HTTP/3 versions;
- WebSocket: Supported natively by nginx without additional user configuration;
- gRPC: Supported; requires applying the annotation;
- FCGI: Supported; requires applying the annotation;
- Proxy Protocol: Support for receiving Proxy Protocol traffic is provided through the use of the corresponding inlets;
- SSL Passthrough: End-to-end SSL traffic routing is supported via the corresponding inlets.

Currently, our `ingress-nginx` implementation does not support TCP/UDP traffic routing (`tcp-services/udp-services` functionality). To route TCP/UDP traffic, we recommend using the native Kubernetes NodePort Service or consider using `alb` module to route traffic using Gateway API approach (available since DKP 1.76 version).

Using HTTPS

The module allows you to manage HTTPS security policies for each Ingress NGINX Controller, including:

- HSTS parameters;
- Available SSL/TLS versions and encryption protocols.

By default controllers of all versions use TLSv1.2 and TLSv1.3 with the option to configure legacy versions. In case of using TLSv1.2 the following protocols are used:

- Key Exchange: ECDHE, DHE;
- Authentication: ECDSA, RSA;
- Encryption (AEAD): AES-GCM (128/256), ChaCha20-Poly1305;
- Integrity (MAC): SHA256, SHA384.

When using TLSv1.3 the available cipher suites are defined by the OpenSSL library (version 3.3.3+).

The `ingress-nginx` module is integrated with with the `cert-manager` module. Thus, it can request SSL certificates automatically and pass them to Ingress NGINX Controllers for further use.

Monitoring and statistics

Our Ingress Nginx implementation has a Prometheus-based system for collecting statistical data built-in. It uses a variety of metrics based on:

- The overall and upstream response time
- Response codes
- Number of repeated requests (retries)
- Request and response sizes
- Request methods
- `content-types`
- Geography of requests, and others

The data can be grouped by the:

- `namespace`
- `vhost`
- `ingress` resource
- `location` (in nginx)

The graphs are collected in convenient dashboards in Grafana, and there is a drill-down option for graphs. For example, when viewing namespace statistics, you can click a link to a dashboard in Grafana and drill down into the `vhosts` statistics in the corresponding `namespace`.

Statistics

Basic principles of collecting statistics

1. The module is called for each request (at the `log_by_lua_block` stage). It calculates the necessary data and forwards it to the buffer (each nginx worker has its own buffer).
2. For every nginx worker at the `init by lua block` stage, the process is run that asynchronously sends data in the `protobuf` format over a tcp socket to `protobuf_exporter` (DKP development) once a second.
3. `protobuf_exporter` runs as a sidecar container in the ingress-controller's Pod. It receives messages in the protobuf format, parses them, aggregates them according to the specified rules, and exports them in the Prometheus format.
4. Every 30 seconds, Prometheus scrapes both the ingress-controller (since it exports some of the required metrics) and protobuf_exporter. Then these data are used for stats.

What information does Prometheus collect, and in what form?

All the collected metrics have service labels that allow you to identify the controller's instance: `controller`, `app`, `instance`, and `endpoint` (displayed in `/prometheus/targets`).

- All metrics (except for geo) exported by protobuf_exporter have three levels of detail:
  - `ingress_nginx_overall_*` — the "bird's-eye view"; all the metrics have `namespace`, `vhost`, and `content_kind` labels attached;
  - `ingress_nginx_detail_*` — — in addition to the `overall` level labels, `ingress`, `service`, `service_port` and `location` are added;

    - `ingress_nginx_detail_backend_*` — some detailed data; they are collected on a per-backend basis. The `pod_ip` label is added to those listed for the detail level.
- The following metrics are collected for the overall and detail levels:
    - `* requests total` — the total number of requests (additional labels: `scheme`, `method`);
    - `* responses total` — the total number of responses (additional labels: `status`);
    - `*_request_seconds_{sum,count,bucket}` — histogram of the response time;
    - `*_bytes_received_{sum,count,bucket}` — histogram of the request size;
    - `* bytes sent {sum,count,bucket}` — histogram of the response size;
    - `*_upstream_response_seconds_{sum,count,bucket}` — histogram of the upstream response time (the sum of the response times of all upstreams is used if several of them are present);
    - `*_lowres_upstream_response_seconds_{sum,count,bucket}` — the same as above but less detailed (it is suitable for visualization, but not at all for calculating quantiles);
    - `*_upstream_retries_{count,sum}` — the number of requests for which retries were sent to backends, and the number of retries;
- The following metrics are collected for the overall level:
    - `*_geohash_total` — the total number of requests with a specific geohash (additional labels: `geohash`, `place`);
- The following metrics are collected for the detail_backend level:
    - `*_lowres_upstream_response_seconds` — same as a similar metric for overall and detail;
    - `*_responses_total` — the total number of responses (additional labels: `status_class` instead of `status`);
    - `*_upstream_bytes_received_sum` — the sum of the backend's response sizes.

Module support and security

The `ingress-nginx` module is covered by Deckhouse Kubernetes Platform maintenance for the entire platform support lifecycle, regardless of the upstream project's development status. The Deckhouse team tracks CVEs in the controller and its dependencies — NGINX, Lua modules, and base images — and delivers fixes in platform releases. For compliance with PCI DSS expectations regarding vendor support and vulnerability remediation timelines, Flant is the responsible vendor of the module.

Third-party components

List of third-party software used in the `ingress-nginx` module:

- Ingress NGINX Controller 1.10.4

  License: Apache License 2.0

  A ingress controller for Kubernetes using NGINX as a reverse proxy and load balancer.

- Ingress NGINX Controller 1.12.1

  License: Apache License 2.0

  A ingress controller for Kubernetes using NGINX as a reverse proxy and load balancer.

- Ingress NGINX Controller 1.14.4

  License: Apache License 2.0

  A ingress controller for Kubernetes using NGINX as a reverse proxy and load balancer.

- nlohmannJson 3.11.3

  License: MIT License

  JSON for Modern C++

- ModSecurity 3.0.14

  License: Apache License 2.0

  ModSecurity is an open source, cross platform web application firewall (WAF) engine for Apache, IIS and Nginx.

- OWASP CRS 4.10.0

  License: Apache License 2.0

  The OWASP CRS is a set of generic attack detection rules for use with ModSecurity or compatible web application firewalls.

- OpenKruise 898dc482a0cbf4fa0db214f952319aa5e1229152

  License: Apache License 2.0

  OpenKruise is an extended component suite for Kubernetes, which mainly focuses on application automations, such as deployment, upgrade, ops and availability protection.

Configuration

> **i** The module lifecycle stage: General Availability

> **i** Pay attention to the global parameter publicDomainTemplate, if you are turning the module on. If the parameter is not specified, the Ingress resources for DKP service components (dashboard, user-auth, grafana, upmeter, etc.) will not be created.

Ingress controllers are configured using the IngressNginxController Custom Resource.

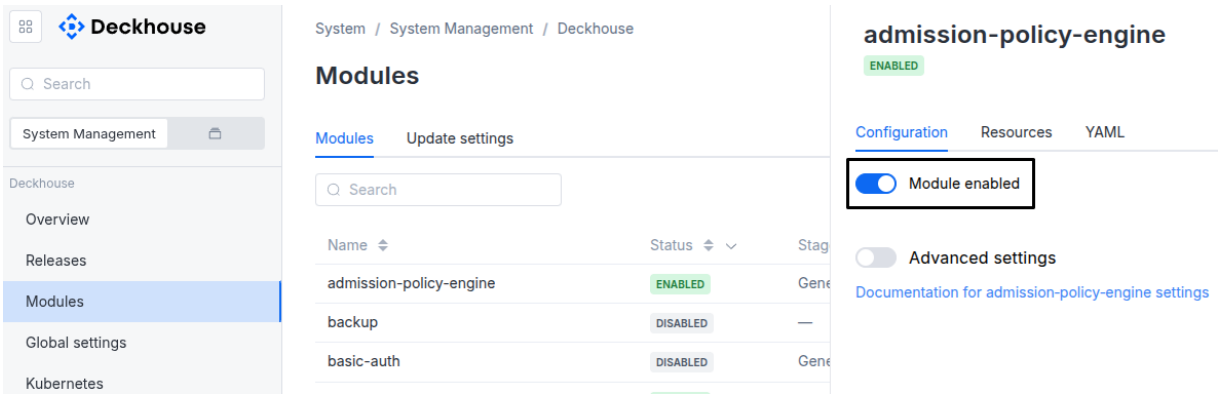
The module has 14 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `ingress-nginx` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable ingress-nginx
  ```

- **Using ModuleConfig `ingress-nginx`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `ingress-nginx` (create it if necessary);

  Example of a manifest to enable module `ingress-nginx`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: ingress-nginx
  spec:
    enabled: true
  ```

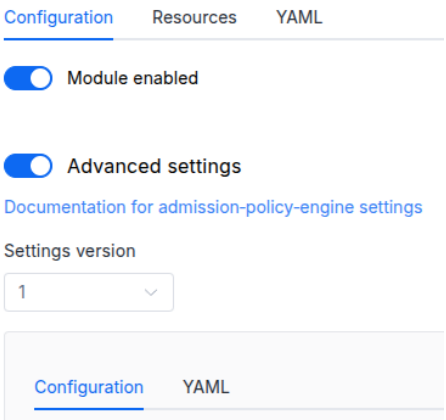
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `ingress-nginx` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `ingress-nginx` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `ingress-nginx`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `ingress-nginx` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc ingress-nginx
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `ingress-nginx` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

Example of a manifest for ModuleConfig `ingress-nginx`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: ingress-nginx
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.defaultControllerVersion**
    string

    The version of the ingress-nginx controller that is used for all controllers by default if the `controllerVersion` parameter is omitted in the IngressNginxController CR.

    Default: `1.10`

    Allowed values: `1.10`, `1.12`, `1.14`
  - **settings.highAvailability**
    boolean

    Manually enable the high availability mode.

    By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

    Example:

    ```
    highAvailability: true
    ```
  - **settings.storageClass**
    string

    The name of the StorageClass to use for storing GeoIP databases (the `geoproxy` StatefulSet).

    If omitted, the StorageClass of the existing PVC is used. If there is no PVC yet, the StorageClass will be chosen according to the global storageClass parameter.

    The global `storageClass` parameter is only considered when the module is enabled. Changing it while the module is enabled will not trigger automatic disk re-provisioning.

    If `false` is specified, `emptyDir` will be used.

    Examples:

    ```
    storageClass: ceph-ssd
    ```

    ```
    storageClass: 'false'
    ```

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

IngressNginxController

Scope: Cluster

- v1
- v1alpha1

- **status**
  object
  - **status.loadBalancer**
    object
    - **status.loadBalancer.hostname**
      string

      Hostname of the load balancer.
    - **status.loadBalancer.ip**
      string

      IP address of the load balancer.

- **spec**
  object

Required value

- **spec.acceptRequestsFrom**
  array of strings

  List of IP addresses in the CIDR format that are allowed accessing the load balancer.

  Regardless of the inlet type, the address to be verified (the `original_address` field in logs) is always the original IP address the connection is established from and not the client address that can be passed in some inlets via headers or using the proxy protocol.

  This parameter is implemented using the map module. If the original address is not allowed, nginx closes the connection, returning the code `444`.

  > **i** By default, the controller can be connected to from any address.

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **spec.additionalHeaders**
  object

  Additional headers to add to all requests. Use the `key: value(string)` format.

- **spec.additionalLogFields**
  object

  Additional fields to add to nginx logs. Use the `key: value(string)` format.

- **spec.annotationValidationEnabled**
  boolean

  Enables validation for Ingress rule annotations.

  Default: `false`

- **spec.chaosMonkey**
  boolean

  Tool for unexpected and random termination of Ingress controller Pods in a systemic manner.

  Intended for testing the fault tolerance of the Ingress controller.

  Default: `false`

- **spec.config**
  object

  Section of the Ingress controller parameters where you can add any supported option in the `key: value(string)` format.

  > **i** An error in options may lead to the failure of the Ingress controller.
  >
  > Be careful when using this parameter, as backward compatibility or stability of the target Ingress controller are not guaranteed.

- **spec.controllerLogLevel**
  string

  Defines the verbosity level for the Ingress controller logs.

  Default: `Info`

  Allowed values: `Error` — Only critical errors will be logged.
  `Warn` — Warnings and errors will be logged.
  `Info` — Informational messages, including basic debugging information.
  `Extended` — Extended information about changes in system state.
  `Debug` — Verbose debugging output, potentially very noisy.
  `Trace` — Trace-level logs with detailed step-by-step context.

- **spec.controllerPodsAdditionalAnnotations**
  object

  Additional custom annotations for Ingress controller pods.

  > **i** Be careful when using this parameter, as backward compatibility or stability of the target Ingress controller are not guaranteed.

- **spec.controllerVersion**
  string

  Ingress NGINX Controller version.

  By default, the version set in the module settings is used.

  Allowed values: `1.10`, `1.12`, `1.14`

- **spec.customErrors**
  object

  Section with parameters for HTTP error customization.

> **i** If this section is defined in the configuration, all its parameters are required.
>
> Changing any parameter causes the restart of all Ingress NGINX Controllers.

- **spec.customErrors.codes**
  array of strings

  Required value

  List of response codes (array) causing redirection to the custom backend.

  - **Element of the array**
    string

    Pattern: `^[1-5][0-9][0-9]$`

- **spec.customErrors.namespace**
  string

  Required value

  Name of a namespace keeping the service used as a default custom backend.

  Example:

  ```
  namespace: default
  ```

- **spec.customErrors.serviceName**
  string

  Required value

  Name of a service to be used as a default custom backend.

  Example:

  ```
  serviceName: custom-errors-backend-service
  ```

- **spec.defaultSSLCertificate**
  object

  The certificate that is used:

  - For `catch-all` server requests (here, “catch-all server” refers to the nginx server directive. Requests without a corresponding Ingress resource end up on the `catch-all` server.
  - For Ingress resources that do not have a `secretName` specified in the `tls` section.

  By default, a self-signed certificate is used.

  > **i** **Caution.** This parameter does not affect certificates used in the Ingress resources of the Deckhouse modules. To specify the certificate to use in the Ingress resources of Deckhouse modules, use the global parameter `modules.https.customCertificate`.

  - **spec.defaultSSLCertificate.secretRef**
    object

    Link to the Secret for passing to the Ingress controller.

    - **spec.defaultSSLCertificate.secretRef.name**
      string

      Name of Secret containing the SSL certificate.

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

    - **spec.defaultSSLCertificate.secretRef.namespace**
      string

      Name of the namespace containing the Secret with the SSL certificate.

      Default: `d8-ingress-nginx`

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **spec.disableHTTP2**
  boolean

  Disables HTTP/2 support.

  Default: `false`

- **spec.enableHTTP3**
  boolean

  Enables HTTP/3 support.

  > **i** HTTP/3 cannot be enabled when the inlet is set to `HostPortWithProxyProtocol` or `LoadBalancerWithProxyProtocol`.
  >
  > For the `HostWithFailover` inlet, HTTP/3 is applied only to the primary controller; the failover controller always runs without HTTP/3.

  Default: `false`

- **spec.enableIstioSidecar**
  boolean

Attaches annotations to the controller pods to automatically inject Istio sidecar containers.

After setting this parameter, the `sidecar.istio.io/inject: "true"` and `traffic.sidecar.istio.io/includeOutboundIPRanges: "<Service CIDR>"` annotations will be attached to the Ingress controller pods. When these pods are created, Istio sidecars will be automatically added to them via a mutating webhook. After that, the sidecar will intercept all traffic directed to the Service CIDR.

To use this feature, add the following annotations to your Ingress resources:

- `nginx.ingress.kubernetes.io/service-upstream: "true"`: Using this annotation, the Ingress controller sends requests to a service ClusterIP (from Service CIDR) instead of sending them to the application pods. The `istio-proxy` sidecar only intercepts traffic directed to Service CIDR, while the remaining requests are sent directly.
- `nginx.ingress.kubernetes.io/upstream-vhost: myservice.myns.svc`: Using this annotation, the sidecar can identify the application service that serves requests.

> **i** **Caution.** This parameter cannot be enabled if the Ingress controller inlet is set to `HostWithFailover`.

- **spec.geoIP2**
  object

  GeoIP2 activation options.

  - **spec.geoIP2.maxmindAccountID**
    integer

    MaxMind Account ID used to authenticate GeoIP2 database downloads.

    Required when `licenseKey` is set. If `accountID` parameter is provided, downloads are performed using the official library `geoipupdate`, which allows you to skip unchanged updates and save the license limit. [https://github.com/maxmind/geoipupdate]

    Allowed values: `1 <= X`

  - **spec.geoIP2.maxmindEditionIDs**
    array of strings

    List of database revisions to download at startup.

    For details on GeoLite databases, refer to the MaxMind blog article.

    Default: `["GeoLite2-City","GeoLite2-ASN"]`

    - **Element of the array**
      string

      Allowed values: `GeoIP2-Anonymous-IP`, `GeoIP2-Country`, `GeoIP2-City`, `GeoIP2-Connection-Type`, `GeoIP2-Domain`, `GeoIP2-ISP`, `GeoIP2-ASN`, `GeoLite2-ASN`, `GeoLite2-Country`, `GeoLite2-City`

  - **spec.geoIP2.maxmindLicenseKey**
    string

    License key to download the GeoIP2 database.

    If the key is set in the configuration, the module downloads the GeoIP2 database every time the controller is started. For details on obtaining a key, refer to the MaxMind blog article.

  - **spec.geoIP2.maxmindMirror**
    object

    Mirror configuration for downloading GeoIP databases. If no parameter is specified or the parameter has an empty value, the default value will be used, which is `https://download.maxmind.com`.

    If this mirror is used to host preloaded archives, the `licenseKey` parameter may be omitted.

    - **spec.geoIP2.maxmindMirror.ca**
      string

      Custom CA certificate in PEM format for verifying TLS connections to the mirror.

      Example: —–BEGIN CERTIFICATE—– … —–END CERTIFICATE—–

    - **spec.geoIP2.maxmindMirror.insecureSkipVerify**
      boolean

      Skip TLS certificate verification for the mirror, for example when using self-signed certificates.

    - **spec.geoIP2.maxmindMirror.url**
      string

      Required value

      The URL for downloading GeoIP databases.

      Pattern: `^https?://.+$`

      Examples:

      ```
      url: https://mirror.local
      ```

      ```
      url: absolute path https://mirror.local/GeoLite2-City.tar.gz
      ```

- **spec.hostPort**
  object

  Section of the `HostPort` inlet parameters.

  - **spec.hostPort.acceptClientIPHeadersFrom**
    array of strings

    Defines a list of trusted CIDR ranges that are known to send correct `X-Forwarded-*/ProxyProtocol` headers.

Default: `["0.0.0.0/0"]`

Example:

```
acceptClientIPHeadersFrom:
- 192.168.0.0/24
```

- **Element of the array**
  string

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **spec.hostPort.behindL7Proxy**
  boolean

  Enables processing and passing of the incoming `X-Forwarded-*` headers.

  > **Caution.** Before using that option, ensure that requests to the Ingress come from trusted sources only. To set restrictions, use the `acceptRequestsFrom` parameter.

- **spec.hostPort.httpPort**
  integer

  Port for insecure HTTP connections.

  If the parameter is not set, HTTP connections cannot be established.

  > This parameter is required if `httpsPort` is not set.

  Example:

  ```
  httpPort: 80
  ```

- **spec.hostPort.httpsPort**
  integer

  Port for secure HTTPS connections.

  If the parameter is not set, HTTPS connections cannot be established.

  > This parameter is required if `httpPort` is not set.

  Example:

  ```
  httpsPort: 443
  ```

- **spec.hostPort.realIPHeader**
  string

  The header for identifying the original IP address of a client.

  > This option works only if `behindL7Proxy` is enabled.

  Default: `X-Forwarded-For`

  Example:

  ```
  realIPHeader: CF-Connecting-IP
  ```

- **spec.hostPortWithProxyProtocol**
  object

  Section of the `HostPortWithProxyProtocol` inlet parameters.

  - **spec.hostPortWithProxyProtocol.acceptClientIPHeadersFrom**
    array of strings

    Defines a list of trusted CIDR ranges that are known to send correct `X-Forwarded-*`/`ProxyProtocol` headers.

    Default: `["0.0.0.0/0"]`

    Example:

    ```
    acceptClientIPHeadersFrom:
    - 192.168.0.0/24
    ```

    - **Element of the array**
      string

      Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  - **spec.hostPortWithProxyProtocol.httpPort**
    integer

    Port for insecure HTTP connections.

If the parameter is not set, HTTP connections cannot be established.

> **i** This parameter is required if `httpsPort` is not set.

Example:

```
httpPort: 80
```

- **spec.hostPortWithProxyProtocol.httpsPort**
  integer

  Port for secure HTTPS connections.

  If the parameter is not set, HTTPS connections cannot be established.

  > **i** This parameter is required if `httpPort` is not set.

  Example:

  ```
  httpsPort: 443
  ```

- **spec.hostPortWithSSLPassthrough**
  object

  Section of the `HostPortWithSSLPassthrough` inlet parameters.

  - **spec.hostPortWithSSLPassthrough.httpPort**
    integer

    Port for insecure HTTP connections.

    If the parameter is not set, HTTP connections cannot be established.

    > **i** This parameter is required if `httpsPort` is not set.

    Example:

    ```
    httpPort: 80
    ```

  - **spec.hostPortWithSSLPassthrough.httpsPort**
    integer

    Port for secure HTTPS connections.

    If the parameter is not set, HTTPS connections cannot be established.

    > **i** This parameter is required if `httpPort` is not set.

    Example:

    ```
    httpsPort: 443
    ```

- **spec.hsts**
  boolean

  Enables HTTP Strict-Transport-Security (HSTS) response headers. For details on HSTS headers, refer to the MDN Web Docs article.

  Default: `false`

- **spec.hstsOptions**
  object

  HSTS parameters.

  - **spec.hstsOptions.includeSubDomains**
    boolean

    Applies HSTS parameters to all subdomains of a website.

    Default: `false`

  - **spec.hstsOptions.maxAge**
    string

    Time in seconds during which the browser remembers that the website is only accessible via HTTPS.

    Default: `31536000`

    Pattern: `^[1-9][0-9]*$`

    Example:

    ```
    maxAge: '31536000'
    ```

  - **spec.hstsOptions.preload**
    boolean

    Adds a website to the preload list.

The list instructs browsers to establish connections to the specified websites over HTTPS only.

Default: `false`

- **spec.ingressClass**
  string

  Name of the Ingress class to use with the Ingress NGINX Controller.

  Using this option, you can create several controllers to use with a single Ingress class.

  > **i** If you set it to `nginx`, Ingress resources without the `kubernetes.io/ingress.class` annotation or the `spec.ingressClassName` field will be handled as well.

  Default: `nginx`

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  ingressClass: nginx
  ```

- **spec.inlet**
  string

  Required value

  The way external traffic is routed to the cluster. Once you set the method, you cannot change it later.

  Allowed values:

  - `LoadBalancer`: The Ingress controller is deployed, and a `LoadBalancer`-type service is provisioned.
  - `LoadBalancerWithProxyProtocol`: The Ingress controller is deployed, and a `LoadBalancer`-type service is provisioned. The Ingress controller uses the proxy-protocol to get a real IP address of the client.

    > **i** **Note.** HTTP/3 (`enableHTTP3`) cannot be enabled with this inlet because QUIC is incompatible with PROXY protocol.

  - `LoadBalancerWithSSLPassthrough`: The Ingress controller is deployed, and a `LoadBalancer`-type service is provisioned. This option enables the SSL Passthrough feature, allowing backends to be configured to accept SSL traffic directly without termination at the Ingress controller.

    > **i** The SSL Passthrough feature is implemented by intercepting all traffic on the configured HTTPS port (default: `443`) and routing it to a local TCP proxy. This implementation bypasses nginx completely and introduces a significant performance penalty.

    The SSL Passthrough protocol leverages Server Name Indication (SNI) and reads the virtual domain data from the TLS handshake protocol, which requires a compatible client.

    If the host name doesn't match the requested host name, the request is forwarded to NGINX on the configured SSL Passthrough proxy port (default: `442`), which subsequently proxies the request to the default backend.

    > **i** Unlike HTTP backends, traffic to SSL Passthrough backends is routed to the ClusterIP of the backing service instead of individual endpoints.
    >
    > Since SSL Passthrough works at layer 4 of the OSI model (TCP) and not layer 7 (HTTP), using SSL Passthrough makes it impossible to use other annotations configured on an Ingress object.

  - `HostPort`: The Ingress controller is deployed and made available on node ports via `hostPort`.
  - `HostPortWithProxyProtocol`: The Ingress controller is deployed and made available on node ports via `hostPort`. The Ingress controller uses the proxy-protocol to get a real IP address of the client.

    > **i** **Caution.** When using this inlet, ensure that requests to the Ingress come from trusted sources only. To enforce it, use the `acceptRequestsFrom` parameter.
    >
    > **Note.** HTTP/3 (`enableHTTP3`) cannot be enabled with this inlet because QUIC is incompatible with PROXY protocol.

  - `HostPortWithSSLPassthrough`: The Ingress controller is deployed and made available on node ports via `hostPort`. This option enables the SSL Passthrough feature, allowing backends to be configured to accept SSL traffic directly without termination at the Ingress controller.

    Specify inlet parameters in the `spec.HostPortWithSSLPassthrough` section.

    The SSL Passthrough protocol leverages SNI and reads the virtual domain data from the TLS handshake protocol, which requires a compatible client.

    If the host name doesn't match the requested host name, the request is forwarded to NGINX on the configured SSL Passthrough proxy port (default: `442`), which subsequently proxies the request to the default backend.

> **i** The SSL Passthrough feature is implemented by intercepting all traffic on the configured HTTPS port (default: `443`) and routing it to a local TCP proxy. This implementation bypasses nginx completely and introduces a significant performance penalty.
>
> Unlike HTTP backends, traffic to SSL Passthrough backends is routed to the ClusterIP of the backing service instead of individual endpoints.
>
> Since SSL Passthrough works at layer 4 of the OSI model (TCP) and not layer 7 (HTTP), using SSL Passthrough makes it impossible to use other annotations configured on an Ingress object.

- `HostWithFailover`: Two Ingress controllers are deployed: a primary and a backup. The primary controller runs in a `hostNetwork`. If the primary controller pods are unavailable, traffic is routed to the backup controller.

  > **i** There can be only one controller with this inlet type on a single host.
  >
  > Ensure the following ports are available on the node: `80`, `81`, `443`, `444`, `4207`, `4208`.
  >
  > To change the inlet, remove the iptables rules and restart the `kube-proxy` pods or reboot the nodes hosting Ingress controllers.
  >
  > This inlet cannot be used if the `enableIstioSidecar` parameter is enabled.
  >
  > **Note.** When HTTP/3 (`enableHTTP3`) is enabled, it is applied only to the primary controller; the failover controller always runs without HTTP/3.

Allowed values: `LoadBalancer`, `LoadBalancerWithSSLPassthrough`, `LoadBalancerWithProxyProtocol`, `HostPort`, `HostPortWithSSLPassthrough`, `HostPortWithProxyProtocol`, `HostWithFailover`

- **spec.legacySSL**
  boolean

  Enables outdated versions of the TLS protocol and cipher suites.

  Enables the following TLS protocol versions: `TLSv1`, `TLSv1.1`, `TLSv1.2`, `TLSv1.3`.

  Enables the following combinations of cipher suites in order from the strongest to the weakest: `ECDHE-ECDSA-AES256-GCM-SHA384`, `ECDHE-RSA-AES256-GCM-SHA384`, `ECDHE-ECDSA-CHACHA20-POLY1305`, `ECDHE-RSA-CHACHA20-POLY1305`, `ECDHE-ECDSA-AES128-GCM-SHA256`, `ECDHE-RSA-AES128-GCM-SHA256`, `ECDHE-ECDSA-AES256-SHA384`, `ECDHE-RSA-AES256-SHA384`, `ECDHE-ECDSA-AES128-SHA256`, `ECDHE-RSA-AES128-SHA256`, `ECDHE-ECDSA-AES256-SHA`, `ECDHE-RSA-AES256-SHA`, `ECDHE-ECDSA-AES128-SHA`, `ECDHE-RSA-AES128-SHA`, `AES256-GCM-SHA384`, `AES128-GCM-SHA256`, `AES256-SHA256`, `AES128-SHA256`, `AES256-SHA`, `AES128-SHA`.

  > **i** By default, only `TLSv1.2` and `TLSv1.3` with the newest cipher suites are enabled.

- **spec.loadBalancer**
  object

  Not required value.

  Section of the `LoadBalancer` inlet parameters.

  - **spec.loadBalancer.acceptClientIPHeadersFrom**
    array of strings

    Defines a list of trusted CIDR ranges that are known to send correct `X-Forwarded-*/ProxyProtocol` headers.

    Default: `["0.0.0.0/0"]`

    Example:

    ```
    acceptClientIPHeadersFrom:
    - 192.168.0.0/24
    ```

    - **Element of the array**
      string

      Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  - **spec.loadBalancer.annotations**
    object

    Annotations to assign to the service for flexible configuration of the load balancer.

    > **i** The module does not take into account the specifics of setting annotations in different clouds.
    >
    > If annotations for provisioning a load balancer are only applied when creating a service, recreate the IngressNginxController resource to update respective parameters (or create a new resource and then delete the old one).

  - **spec.loadBalancer.behindL7Proxy**
    boolean

    Enables processing and passing of the incoming `X-Forwarded-*` headers.

    > **i** **Caution.** Before using that option, ensure that requests to the Ingress come from trusted sources only.

  - **spec.loadBalancer.httpPort**
    integer

External port for insecure HTTP connections exposed by the provisioned `LoadBalancer` service.

If neither `httpPort` nor `httpsPort` is set, the service exposes the standard ports `80` and `443`. If at least one of these parameters is specified, the service will listen only to explicitly specified ports for the corresponding connection type. For connection types for which no parameter is specified, nothing changes: the standard port is listened to.

> **Caution.** Changing service ports may trigger provider-specific side effects (for example, recreating the LoadBalancer or changing its public IP address). Such behavior depends on the cloud implementation; test it before using it in production.
>
> **Caution.** ACME HTTP-01 challenges (e.g. Let's Encrypt via cert-manager) require the Ingress controller to be reachable on TCP port `80`. If you set `httpsPort` without `httpPort`, or set `httpPort` to a value other than `80`, certificate issuance/renewal may stop working unless you use DNS-01 or forward port `80` to the controller externally.
>
> **Caution.** Many applications (including some Deckhouse module components) form absolute URLs and redirects without an explicit port (assuming standard ports `80/443`). If you expose the Ingress controller on non-standard external ports, users may be redirected to the wrong port and links may be generated incorrectly; prefer using standard ports externally or configure external port forwarding/proxying.

Default: `80`

Allowed values: `1 <= X <= 65535`

- **spec.loadBalancer.httpsPort**
  integer

  External port for secure HTTPS connections exposed by the provisioned `LoadBalancer` service.

  If neither `httpPort` nor `httpsPort` is set, the service exposes the standard ports `80` and `443`. If at least one of these parameters is specified, the service will listen only to explicitly specified ports for the corresponding connection type. For connection types for which no parameter is specified, nothing changes: the standard port is listened to.

  > **Caution.** Changing service ports may trigger provider-specific side effects (for example, recreating the LoadBalancer or changing its public IP address). Such behavior depends on the cloud implementation; test it before using it in production.
  >
  > **Caution.** Many applications (including some Deckhouse module components) form absolute URLs and redirects without an explicit port (assuming standard ports `80/443`). If you set `httpsPort` to a non-standard value, users may be redirected to TCP port `443` and links may be generated incorrectly; prefer using standard ports externally or configure external port forwarding/proxying.

  Default: `443`

  Allowed values: `1 <= X <= 65535`

- **spec.loadBalancer.loadBalancerClass**
  string

  Class of the load balancer for incoming network requests (passed to the `spec.loadBalancerClass` parameter of the provisioned service with the `LoadBalancer` type).

- **spec.loadBalancer.realIPHeader**
  string

  The header for identifying the original IP address of a client.

  > This option works only if `behindL7Proxy` is enabled.

  Default: `X-Forwarded-For`

  Example:

  ```
  realIPHeader: CF-Connecting-IP
  ```

- **spec.loadBalancer.sourceRanges**
  array of strings

  List of IP addresses in the CIDR format that are allowed accessing the load balancer.

  > **Caution.** A cloud provider may not support this option or ignore it.
  >
  > Providers supporting this option: AWS, Azure, GCP. Providers ignoring this option: Yandex Cloud.
  >
  > For other cloud providers, behavior may vary depending on the cloud implementation specifics. Test this option before using it in a production environment.

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **spec.loadBalancerWithProxyProtocol**
  object

  Not required value.

  Section of the `LoadBalancerWithProxyProtocol` inlet parameters.

  - **spec.loadBalancerWithProxyProtocol.acceptClientIPHeadersFrom**
    array of strings

Defines a list of trusted CIDR ranges that are known to send correct `X-Forwarded-*`/`ProxyProtocol` headers.

Default: `["0.0.0.0/0"]`

Example:

```
acceptClientIPHeadersFrom:
- 192.168.0.0/24
```

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **spec.loadBalancerWithProxyProtocol.annotations**
  object

  Annotations to assign to the service for flexible configuration of the load balancer.

  > **i**
  >
  > The module does not take into account the specifics of setting annotations in different clouds.
  >
  > If annotations for provisioning a load balancer are only applied when creating a service, recreate the IngressNginxController resource to update respective parameters (or create a new resource and then delete the old one).

- **spec.loadBalancerWithProxyProtocol.httpPort**
  integer

  External port for insecure HTTP connections exposed by the provisioned `LoadBalancer` service.

  If neither `httpPort` nor `httpsPort` is set, the service exposes the standard ports `80` and `443`. If at least one of these parameters is specified, the service will listen only to explicitly specified ports for the corresponding connection type. For connection types for which no parameter is specified, nothing changes: the standard port is listened to.

  > **i**
  >
  > **Caution.** Changing service ports may trigger provider-specific side effects (for example, recreating the LoadBalancer or changing its public IP address). Such behavior depends on the cloud implementation; test it before using it in production.
  >
  > **Caution.** ACME HTTP-01 challenges (e.g. Let’s Encrypt via cert-manager) require the Ingress controller to be reachable on TCP port `80`. If you set `httpsPort` without `httpPort`, or set `httpPort` to a value other than `80`, certificate issuance/renewal may stop working unless you use DNS-01 or forward port `80` to the controller externally.
  >
  > **Caution.** Many applications (including some Deckhouse module components) form absolute URLs and redirects without an explicit port (assuming standard ports `80`/`443`). If you expose the Ingress controller on non-standard external ports, users may be redirected to the wrong port and links may be generated incorrectly; prefer using standard ports externally or configure external port forwarding/proxying.

  Default: `80`

  Allowed values: `1 <= X <= 65535`

- **spec.loadBalancerWithProxyProtocol.httpsPort**
  integer

  External port for secure HTTPS connections exposed by the provisioned `LoadBalancer` service.

  If neither `httpPort` nor `httpsPort` is set, the service exposes the standard ports `80` and `443`. If at least one of these parameters is specified, the service will listen only to explicitly specified ports for the corresponding connection type. For connection types for which no parameter is specified, nothing changes: the standard port is listened to.

  > **i**
  >
  > **Caution.** Changing service ports may trigger provider-specific side effects (for example, recreating the LoadBalancer or changing its public IP address). Such behavior depends on the cloud implementation; test it before using it in production.
  >
  > **Caution.** Many applications (including some Deckhouse module components) form absolute URLs and redirects without an explicit port (assuming standard ports `80`/`443`). If you set `httpsPort` to a non-standard value, users may be redirected to TCP port `443` and links may be generated incorrectly; prefer using standard ports externally or configure external port forwarding/proxying.

  Default: `443`

  Allowed values: `1 <= X <= 65535`

- **spec.loadBalancerWithProxyProtocol.loadBalancerClass**
  string

  Class of the load balancer for incoming network requests (passed to the `spec.loadBalancerClass` parameter of the provisioned service with the `LoadBalancer` type).

- **spec.loadBalancerWithProxyProtocol.sourceRanges**
  array of strings

  List of IP addresses in the CIDR format that are allowed accessing the load balancer.

> **Caution.** A cloud provider may not support this option or ignore it.
>
> Providers supporting this option: AWS, Azure, GCP. Providers ignoring this option: Yandex Cloud.
>
> For other cloud providers, behavior may vary depending on the cloud implementation specifics. Test this option before using it in a production environment.

- **Element of the array**
  string

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **spec.loadBalancerWithSSLPassthrough**
  object

  Not required value.

  Section of the `LoadBalancerWithSSLPassthrough` inlet parameters.

  - **spec.loadBalancerWithSSLPassthrough.annotations**
    object

    Annotations to assign to the service for flexible configuration of the load balancer.

    > The module does not take into account the specifics of setting annotations in different clouds.
    >
    > If annotations for provisioning a load balancer are only applied when creating a service, recreate the IngressNginxController resource to update respective parameters (or create a new resource and then delete the old one).

  - **spec.loadBalancerWithSSLPassthrough.httpPort**
    integer

    External port for insecure HTTP connections exposed by the provisioned `LoadBalancer` service.

    If neither `httpPort` nor `httpsPort` is set, the service exposes the standard ports `80` and `443`. If at least one of these parameters is specified, the service will listen only to explicitly specified ports for the corresponding connection type. For connection types for which no parameter is specified, nothing changes: the standard port is listened to.

    > **Caution.** Changing service ports may trigger provider-specific side effects (for example, recreating the LoadBalancer or changing its public IP address). Such behavior depends on the cloud implementation; test it before using it in production.
    >
    > **Caution.** ACME HTTP-01 challenges (e.g. Let’s Encrypt via cert-manager) require the Ingress controller to be reachable on TCP port `80`. If you set `httpsPort` without `httpPort`, or set `httpPort` to a value other than `80`, certificate issuance/renewal may stop working unless you use DNS-01 or forward port `80` to the controller externally.
    >
    > **Caution.** Many applications (including some Deckhouse module components) form absolute URLs and redirects without an explicit port (assuming standard ports `80`/`443`). If you expose the Ingress controller on non-standard external ports, users may be redirected to the wrong port and links may be generated incorrectly; prefer using standard ports externally or configure external port forwarding/proxying.

    Default: `80`

    Allowed values: `1 <= X <= 65535`

  - **spec.loadBalancerWithSSLPassthrough.httpsPort**
    integer

    External port for secure HTTPS connections exposed by the provisioned `LoadBalancer` service.

    If neither `httpPort` nor `httpsPort` is set, the service exposes the standard ports `80` and `443`. If at least one of these parameters is specified, the service will listen only to explicitly specified ports for the corresponding connection type. For connection types for which no parameter is specified, nothing changes: the standard port is listened to.

    > **Caution.** Changing service ports may trigger provider-specific side effects (for example, recreating the LoadBalancer or changing its public IP address). Such behavior depends on the cloud implementation; test it before using it in production.
    >
    > **Caution.** Many applications (including some Deckhouse module components) form absolute URLs and redirects without an explicit port (assuming standard ports `80`/`443`). If you set `httpsPort` to a non-standard value, users may be redirected to TCP port `443` and links may be generated incorrectly; prefer using standard ports externally or configure external port forwarding/proxying.

    Default: `443`

    Allowed values: `1 <= X <= 65535`

  - **spec.loadBalancerWithSSLPassthrough.loadBalancerClass**
    string

    Class of the load balancer for incoming network requests (passed to the `spec.loadBalancerClass` parameter of the provisioned service with the `LoadBalancer` type).

  - **spec.loadBalancerWithSSLPassthrough.sourceRanges**
    array of strings

    List of IP addresses in the CIDR format that are allowed accessing the load balancer.

> **Caution.** A cloud provider may not support this option or ignore it.
>
> Providers supporting this option: AWS, Azure, GCP. Providers ignoring this option: Yandex Cloud.
>
> For other cloud providers, behavior may vary depending on the cloud implementation specifics. Test this option before using it in a production environment.

- **Element of the array**
  string

  Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **spec.maxReplicas**
  integer

  Maximum number of `LoadBalancer`, `LoadBalancerWithProxyProtocol` and `LoadBalancerWithSSLPassthrough` replicas for HPA.

  Default: `1`

  Allowed values: `1 <= X`

- **spec.minReplicas**
  integer

  Minimum number of `LoadBalancer`, `LoadBalancerWithProxyProtocol` and `LoadBalancerWithSSLPassthrough` replicas for HPA.

  Default: `1`

  Allowed values: `1 <= X`

- **spec.nginxProfilingEnabled**
  boolean

  Enables NGINX profiling using Valgrind. A memory analysis report is saved to the `/var/log/valgrind` directory on the host. When enabled, Valgrind creates a file named `memcheck.${timestamp}.log`.

  After debugging is complete, don’t forget to disable profiling mode!

  The report contains information about the process behavior, including potential memory leaks.

  > **Caution**. Enabling this option significantly increases memory usage and slows down HTTP/HTTPS request handling by the NGINX process.

  > **Be aware.** When the option is enabled, the ingress-nginx controller pod starts in privileged mode.

- **spec.nodeSelector**
  object

  Same as the `spec.nodeSelector` parameter in Kubernetes Pods.

  If the parameter is omitted or set to `false`, Deckhouse will try to determine the value automatically.

  Use the format of a standard `nodeSelector` list. Instance pods inherit this field as is.

- **spec.resourcesRequests**
  object

  Maximum amount of CPU and memory resources a Pod can request when selecting a node. If the VPA is disabled, these max values become the default ones.

  - **spec.resourcesRequests.mode**
    string

    Required value

    Resource request management mode.

    Default: `VPA`

    Allowed values: `VPA`, `Static`

  - **spec.resourcesRequests.static**
    object

    Static management mode parameters.

    - **spec.resourcesRequests.static.cpu**
      string

      Value for CPU requests.

      Default: `350m`

    - **spec.resourcesRequests.static.memory**
      string

      Value for memory requests.

      Default: `500Mi`

  - **spec.resourcesRequests.vpa**
    object

    Vertical Pod Autoscaler (VPA) mode parameters.

- **spec.resourcesRequests.vpa.cpu**
  object

  Parameters of CPU request restrictions.

  - **spec.resourcesRequests.vpa.cpu.max**
    string

    Maximum value of allowed CPU requests to be submitted by the VPA.

    Default: `50m`

  - **spec.resourcesRequests.vpa.cpu.min**
    string

    Minimum value of allowed CPU requests to be submitted by the VPA.

    Default: `10m`

- **spec.resourcesRequests.vpa.memory**
  object

  Parameters of memory request restrictions.

  - **spec.resourcesRequests.vpa.memory.max**
    string

    Maximum value of allowed memory requests to be submitted by the VPA.

    Default: `200Mi`

  - **spec.resourcesRequests.vpa.memory.min**
    string

    Minimum value of allowed memory requests to be submitted by the VPA.

    Default: `50Mi`

- **spec.resourcesRequests.vpa.mode**
  string

  VPA usage mode.

  Default: `Initial`

  Allowed values: `Initial`, `Auto`

- **spec.tolerations**
  array of objects

  Same as the `spec.tolerations` parameter in Kubernetes Pods.

  If the parameter is omitted or set to `false`, all the possible tolerations are automatically applied the module's Pods.

  Use the format of a standard toleration list. Instance pods inherit this field as is.

  - **spec.tolerations.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **spec.tolerations.key**
    string
  - **spec.tolerations.operator**
    string

    Default: `Equal`

    Allowed values: `Exists`, `Equal`

  - **spec.tolerations.tolerationSeconds**
    integer
  - **spec.tolerations.value**
    string
- **spec.underscoresInHeaders**
  boolean

  Enables using the underscore symbol in headers.

  Related resources:

  - On using the underscores in headers.
  - Why you should be careful when enabling underscores in headers .

  Default: `false`

- **spec.validationEnabled**
  boolean

  Enables validation for Ingress rules.

  > **i** **Warning.** Enabling validation increases the load on the master nodes of the cluster.

  Default: `true`

- **spec.waitLoadBalancerOnTerminating**
  integer

  Number of seconds before the `/healthz` endpoint begins returning the `500` code when the pod enters the `Terminating` state.

Deprecated resource. Support for the resource might be removed in a later release.

- **spec**
  object

  Required value

  - **spec.acceptRequestsFrom**
    array of strings

    IP or CIDR that is allowed to access the Ingress controller.

    Regardless of the inlet type, the source IP address gets always verified (the `original_address` field in logs) (the address that the connection was established from) and not the “address of the client” that can be passed in some inlets via headers or using the proxy protocol.

    This parameter is implemented using the map module. If the source address is not in the list of allowed addresses, nginx closes the connection immediately using HTTP code 444.

    By default, the connection to the controller can be made from any address.

    - **Element of the array**
      string

      Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

  - **spec.additionalHeaders**
    object

    Additional headers to add to all request. (map: key (string)).

  - **spec.additionalLogFields**
    object

    Additional fields to add to nginx logs. (map: key (string)).

  - **spec.annotationValidationEnabled**
    boolean

    Enables the annotation validation feature.

    Default: `false`

  - **spec.chaosMonkey**
    boolean

    The instrument for unexpected and random termination of ingress controller Pods in a systemic manner. Chaos Monkey tests the resilience of ingress controller.

    Default: `false`

  - **spec.config**
    object

    The section with the Ingress controller parameters.

    You can specify any supported parameter in it in the `key: value (string)` format.

    > **i** **Caution.** An erroneous option may lead to the failure of the ingress controller;

    > **i** **Caution.** The usage of this parameter is not recommended; the backward compatibility or operability of the ingress controller that uses this option is not guaranteed

  - **spec.controllerVersion**
    string

    One of the supported Ingress NGINX Controller versions.

    By default: the version in the module settings is used.

    Allowed values: `1.10`, `1.12`, `1.14`

  - **spec.customErrors**
    object

    The section with parameters of custom HTTP errors.

    All parameters in this section are mandatory if it is defined. Changing any parameter **leads to the restart of all Ingress NGINX Controllers**.

    - **spec.customErrors.codes**
      array of strings

      Required value

      Error codes which should be redirected to custom errors backend.

      - **Element of the array**
        string

        Pattern: `^[1-5][0-9][0-9]$`

    - **spec.customErrors.namespace**
      string

      Required value

      Namespace of custom errors backend.

      Example:

```
namespace: default
```

- **spec.customErrors.serviceName**
  string

  Required value

  Name of kubernetes service that leads to custom errors backend.

  Example:

  ```
  serviceName: custom-errors-backend-service
  ```

- **spec.defaultSSLCertificate**
  object

  This certificate is used:

  - for `catch-all` server requests (here, “catch-all server” refers to the nginx server directive. Requests for which there is no corresponding Ingress resource end up on the `catch-all` server.
  - for Ingress resources that do not have a `secretName` specified in the `tls` section.

  By default, a self-signed certificate is used.

  > **Caution.** This parameter does not affect certificates used in the Ingress resources of the Deckhouse modules. You can specify the certificate to be used in the Ingress resources of the Deckhouse modules with the modules.https.customCertificate global parameter.

  - **spec.defaultSSLCertificate.secretRef**
    object

    The Secret reference to pass to the Ingress Controller.

    - **spec.defaultSSLCertificate.secretRef.name**
      string

      Name of Secret containing SSL—certificate.

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

    - **spec.defaultSSLCertificate.secretRef.namespace**
      string

      Namespace, where the Secret is located.

      Default: `d8-ingress-nginx`

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **spec.disableHTTP2**
  boolean

  Switch off HTTP2 support.

  Default: `false`

- **spec.enableIstioSidecar**
  boolean

  Attach annotations to the controller pods to automatically inject Istio sidecar containers.

  After setting this parameter, the `sidecar.istio.io/inject: "true"` and `traffic.sidecar.istio.io/includeOutboundIPRanges: "<Service CIDR>"` annotations will be attached to the ingress-controller pods. During pod creation, the Istio’s mutating webhook will add the sidecar to it. After that, the sidecar will catch the network traffic to Service CIDR.

  To use this feature in your application, you must add these annotations to your Ingress resources:

  - `nginx.ingress.kubernetes.io/service-upstream: "true"` — using this annotation, the ingress-controller sends requests to a single ClusterIP (from Service CIDR) while envoy load balances them. Istio sidecar containers only catching traffic directed to Service CIDR.
  - `nginx.ingress.kubernetes.io/upstream-vhost: myservice.myns.svc` — using this annotation, the sidecar can identify the application service that serves requests.

  > **Caution.** This parameter cannot be enabled if the Ingress controller inlet is set to `HostWithFailover`.

- **spec.geoIP2**
  object

  Enable GeoIP2 databases.

  - **spec.geoIP2.maxmindEditionIDs**
    array of strings

    A list of database editions to download at startup.

    More info...

    Default: `["GeoLite2-City","GeoLite2-ASN"]`

    - **Element of the array**
      string

      Allowed values: `GeoIP2-Anonymous-IP`, `GeoIP2-Country`, `GeoIP2-City`, `GeoIP2-Connection-Type`, `GeoIP2-Domain`, `GeoIP2-ISP`, `GeoIP2-ASN`, `GeoLite2-ASN`, `GeoLite2-Country`, `GeoLite2-City`

  - **spec.geoIP2.maxmindLicenseKey**

string

A license key to download the GeoIP2 database.

If the key is set, the module downloads the GeoIP2 database every time the controller is started. Click here to learn more about obtaining a license key.

- **spec.hostPort**
  object

  `HostPort` inlet settings.

  - **spec.hostPort.behindL7Proxy**
    boolean

    Accepts all the incoming X-Forwarded-* headers and passes them to upstreams.

    > **Caution.** Make sure that requests to the ingress are sent from trusted sources when using this option. The `acceptRequestsFrom` parameter can help you with defining trusted sources.

  - **spec.hostPort.httpPort**
    integer

    Port for insecure HTTP connections.

    If the parameter is not set, the connection over HTTP cannot be established.

    This parameter is mandatory if `httpsPort` is not set.

    Example:

    ```
    httpPort: 80
    ```

  - **spec.hostPort.httpsPort**
    integer

    Port for secure HTTPS connections.

    If the parameter is not set, the connection over HTTPS cannot be established.

    This parameter is mandatory if `httpPort` is not set.

    Example:

    ```
    httpsPort: 443
    ```

  - **spec.hostPort.realIPHeader**
    string

    Sets the header field for identifying the originating IP address of a client.

    This option works only if `behindL7Proxy` is enabled.

    Default: `X-Forwarded-For`

    Example:

    ```
    realIPHeader: CF-Connecting-IP
    ```

- **spec.hostPortWithProxyProtocol**
  object

  A section of parameters of the `HostPortWithProxyProtocol` inlet.

  - **spec.hostPortWithProxyProtocol.httpPort**
    integer

    Port for insecure HTTP connections.

    If the parameter is not set, the connection over HTTP cannot be established.

    This parameter is mandatory if `httpsPort` is not set.

    Example:

    ```
    httpPort: 80
    ```

  - **spec.hostPortWithProxyProtocol.httpsPort**
    integer

    Port for secure HTTPS connections.

    If the parameter is not set, the connection over HTTPS cannot be established.

    This parameter is mandatory if `httpPort` is not set.

    Example:

    ```
    httpsPort: 443
    ```

- **spec.hsts**
  boolean

  Determines whether hsts is enabled. Read more...).

  Default: `false`

- **spec.hstsOptions**
  object

  Options for HTTP Strict Transport Security.

  - **spec.hstsOptions.includeSubDomains**
    boolean

    If this optional parameter is specified, this rule applies to all of subdomains as well.

    Default: `false`

  - **spec.hstsOptions.maxAge**
    string

    The time, in seconds, that the browser should remember that a site is only to be accessed using HTTPS.

    Default: `31536000`

    Pattern: `^[1-9][0-9]*$`

    Example:

    ```
    maxAge: '31536000'
    ```

  - **spec.hstsOptions.preload**
    boolean

    Add your site to preload list to enforce to use SSL/TLS connections on your site.

    Default: `false`

- **spec.ingressClass**
  string

  The name of the Ingress class to use with the Ingress NGINX Controller.

  Using this option, you can create several controllers to use with a single ingress

  > **Caution.** If you set it to “nginx”, then Ingress resources lacking the `kubernetes.io/ingress.class` annotation or `spec.ingressClassName` field will also be handled.

  Default: `nginx`

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  ingressClass: nginx
  ```

- **spec.inlet**
  string

  Required value

  The way traffic from the external network is routed to the cluster. Once you have set the method, you cannot change it later.

  - `LoadBalancer` — Ingress controller is deployed and the service of `LoadBalancer` type is provisioned.
  - `LoadBalancerWithProxyProtocol` — Ingress controller is deployed and the service of `LoadBalancer` type is provisioned. Ingress controller uses proxy-protocol to get a real IP of the client.
  - `HostPort` — Ingress controller is deployed and available through nodes’ ports via `hostPort`;

    Settings are required in spec.HostPort.

  - `HostPortWithProxyProtocol` — Ingress controller is deployed and available through nodes’ ports via `hostPort, it uses proxy-protocol to get a real IP of the client;

    Settings are required in spec.HostPortWithProxyProtocol.

    > **Caution.** Make sure that requests to the Ingress are sent from trusted sources when using this inlet. The acceptRequestsFrom parameter can help you with defining trusted sources.

  - `HostWithFailover` — installs two ingress controllers, the primary and the backup one. The primary controller runs in a hostNetwork. If the pods of the primary controller are not available, the traffic is routed to the backup one;

    > **Caution.** There can be only one controller with this inlet type on a host.

    > **Caution.** The following ports must be available on the node: 80, 81, 443, 444, 4207, 4208.

    > **Caution.** To change inlet, remove the iptables rules and restart the `kube-proxy` pods or reboot the nodes hosting Ingress controllers.

    > **Caution.** This inlet cannot be used if the `enableIstioSidecar` parameter is enabled.

  Allowed values: `LoadBalancer`, `LoadBalancerWithProxyProtocol`, `HostPort`, `HostPortWithProxyProtocol`, `HostWithFailover`

- **spec.legacySSL**
  boolean

  Enable outdated versions of the TLS protocol and cipher suites.

  Enables the following TLS protocol versions: `TLSv1`, `TLSv1.1`, `TLSv1.2`, `TLSv1.3`.

  Enables the following combinations of cipher suites in order from the strongest to the weakest: `ECDHE-ECDSA-AES256-GCM-SHA384`, `ECDHE-RSA-AES256-GCM-SHA384`, `ECDHE-ECDSA-CHACHA20-POLY1305`, `ECDHE-RSA-CHACHA20-POLY1305`, `ECDHE-ECDSA-AES128-GCM-SHA256`, `ECDHE-RSA-AES128-GCM-SHA256`, `ECDHE-ECDSA-AES256-SHA384`, `ECDHE-RSA-AES256-SHA384`, `ECDHE-ECDSA-AES128-SHA256`, `ECDHE-RSA-AES128-SHA256`, `ECDHE-ECDSA-AES256-SHA`, `ECDHE-RSA-AES256-SHA`, `ECDHE-ECDSA-AES128-SHA`, `ECDHE-RSA-AES128-SHA`, `AES256-GCM-SHA384`, `AES128-GCM-SHA256`, `AES256-SHA256`, `AES128-SHA256`, `AES256-SHA`, `AES128-SHA`.

  By default, only `TLSv1.2` and `TLSv1.3` with the newest cipher suites are enabled.

- **spec.loadBalancer**
  object

  Not required value.

  A section of parameters of the `LoadBalancer` inlet.

  - **spec.loadBalancer.annotations**
    object

    Annotations to assign to the service for flexible configuration of the load balancer.

    > **Caution.** The module does not take into account the specifics of setting annotations in different clouds. Note that you will need to recreate `IngressNginxController` (or create a new controller and then delete the old one) if annotations to provision a load balancer are only used when creating the service.

  - **spec.loadBalancer.behindL7Proxy**
    boolean

    Accepts all the incoming `X-Forwarded-*` headers and passes them to upstreams.

    > **Caution.** Make sure that requests to the Ingress controller are sent from trusted sources when using this option.

  - **spec.loadBalancer.loadBalancerClass**
    string

    The LoadBalancer class (it is passed to the `spec.loadBalancerClass` parameter of the provisioned service of the LoadBalancer type).

  - **spec.loadBalancer.realIPHeader**
    string

    Sets the header field for identifying the originating IP address of a client.

    This option works only if `behindL7Proxy` is enabled.

    Default: `X-Forwarded-For`

    Example:

    ```
    realIPHeader: CF-Connecting-IP
    ```

  - **spec.loadBalancer.sourceRanges**
    array of strings

    IP ranges (CIDR) that are allowed to access the load balancer.

    > **Caution.** The cloud provider may not support this option or ignore it. Providers supporting the option: *AWS*, *GCP*, *Azure*. Providers ignoring the option: *YandexCloud*. For other cloud providers, the behavior may depend on the specifics of the cloud implementation. Testing is recommended before use of the option in production.

    - **Element of the array**
      string

      Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **spec.loadBalancerWithProxyProtocol**
  object

  Not required value.

  A section of parameters of the `LoadBalancerWithProxyProtocol` inlet.

  - **spec.loadBalancerWithProxyProtocol.annotations**
    object

    Annotations that will be passed to service with type load balancer to configure it.

    > **Caution.** The module does not take into account the specifics of setting annotations in different clouds. Note that you will need to recreate `IngressNginxController` (or create a new controller and then delete the old one) if annotations to provision a load balancer are only used when creating the service.

  - **spec.loadBalancerWithProxyProtocol.loadBalancerClass**
    string

    The LoadBalancer class (it is passed to the `spec.loadBalancerClass` parameter of the provisioned service of the

LoadBalancer type).

- **spec.loadBalancerWithProxyProtocol.sourceRanges**
  array of strings

  IP ranges (CIDR) that are allowed to access the load balancer.

  > **Caution.** The cloud provider may not support this option or ignore it. Providers supporting the option: *AWS*, *GCP*, *Azure*. Providers ignoring the option: *YandexCloud*. For other cloud providers, the behavior may depend on the specifics of the cloud implementation. Testing is recommended before use of the option in production.

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))$`

- **spec.maxReplicas**
  integer

  LoadBalancer and LoadBalancerWithProxyProtocol controller's Horizontal Pod Autoscaler maximum replicas count.

  Default: `1`

  Allowed values: `1 <= X`

- **spec.minReplicas**
  integer

  LoadBalancer and LoadBalancerWithProxyProtocol controller's Horizontal Pod Autoscaler minimum replicas count.

  Default: `1`

  Allowed values: `1 <= X`

- **spec.nodeSelector**
  object

  The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Format: the standard `nodeSelector` list. Instance pods inherit this field as is.

- **spec.resourcesRequests**
  object

  Max amounts of CPU and memory resources that the pod can request when selecting a node (if the VPA is disabled, then these values become the default ones).

  - **spec.resourcesRequests.mode**
    string

    Required value

    The mode for managing resource requests.

    Default: `VPA`

    Allowed values: `VPA`, `Static`

  - **spec.resourcesRequests.static**
    object

    Static mode settings.

    - **spec.resourcesRequests.static.cpu**
      string

      CPU requests.

      Default: `350m`

    - **spec.resourcesRequests.static.memory**
      string

      Memory requests.

      Default: `500Mi`

  - **spec.resourcesRequests.vpa**
    object

    Parameters of the vpa mode.

    - **spec.resourcesRequests.vpa.cpu**
      object

      CPU-related parameters.

      - **spec.resourcesRequests.vpa.cpu.max**
        string

        Maximum allowed CPU requests.

        Default: `50m`

      - **spec.resourcesRequests.vpa.cpu.min**
        string

        Minimum allowed CPU requests.

Default: `10m`

- **spec.resourcesRequests.vpa.memory**
  object

  The amount of memory requested.

  - **spec.resourcesRequests.vpa.memory.max**
    string

    Maximum allowed memory requests.

    Default: `200Mi`

  - **spec.resourcesRequests.vpa.memory.min**
    string

    Minimum allowed memory requests.

    Default: `50Mi`

- **spec.resourcesRequests.vpa.mode**
  string

  The VPA usage mode.

  Default: `Initial`

  Allowed values: `Initial`, `Auto`

- **spec.tolerations**
  array of objects

  The same as in the pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  Format: the standard toleration list. Instance pods inherit this field as is.

  - **spec.tolerations.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **spec.tolerations.key**
    string
  - **spec.tolerations.operator**
    string

    Default: `Equal`

    Allowed values: `Exists`, `Equal`

  - **spec.tolerations.tolerationSeconds**
    integer
  - **spec.tolerations.value**
    string
- **spec.underscoresInHeaders**
  boolean

  Determines whether underscores are allowed in headers. Learn more....

  This tutorial sheds light on why you should not enable it without careful consideration.

  Default: `false`

- **spec.validationEnabled**
  boolean

  Enable ingress validation admission.

  Default: `true`

- **spec.waitLoadBalancerOnTerminating**
  integer

  The number of seconds before the /healthz location will start to return a 500 code when the pod enters the Terminating state. This parameter has default values:

  - 0s - for HostWithFailover
  - 60s - for HostPort and HostPortWithProxyProtocol
  - 120s - for LoadBalancer and LoadBalancerWithProxyProtocol

FAQ

> **i** The module lifecycle stage: General Availability

How do I limit access to the application in the cluster to Ingress controllers only?

If you need to limit access to your application inside the cluster exclusively from Ingress Pods, you should add a kube-rbac-proxy container to the application Pod as shown in the example below:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  namespace: my-namespace
spec:
  selector:
    matchLabels:
      app: my-app
  replicas: 1
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-sa
      containers:
      - name: my-cool-app
        image: mycompany/my-app:v0.5.3
        args:
        - "--listen=127.0.0.1:8080"
        livenessProbe:
          httpGet:
            path: /healthz
            port: 443
            scheme: HTTPS
      - name: kube-rbac-proxy
        image: flant/kube-rbac-proxy:v0.1.0
        # It is recommended to use a proxy from Deckhouse repository.
        args:
        - "--secure-listen-address=0.0.0.0:443"
        - "--config-file=/etc/kube-rbac-proxy/config-file.yaml"
        - "--v=2"
        - "--logtostderr=true"
        # The user authentication and authorization are not possible if the kube-apiserver is not available.
        # Stale Cache stores the results of successful authorization and is used only if the apiserver is not
available.
        - "--stale-cache-interval=1h30m"
        ports:
        - containerPort: 443
          name: https
        volumeMounts:
        - name: kube-rbac-proxy
          mountPath: /etc/kube-rbac-proxy
      volumes:
      - name: kube-rbac-proxy
        configMap:
          name: kube-rbac-proxy
```

The application only accepts localhost (127.0.0.1) requests. That means that an unsecured connection can only be established to it from within the Pod. At the same time, the proxy listens on 0.0.0.0 and intercepts all external traffic to the Pod.

How do I provide minimum rights to the Service Account?

The proxy needs permissions to create `TokenReview` and `SubjectAccessReview` to authenticate and authorize users using the kube-apiserver.

DKP clusters already have a ready-made ClusterRole — **d8-rbac-proxy**, you don’t need to create it yourself! Link it to your Deployment’s ServiceAccount, as shown in the example below.

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-sa
  namespace: my-namespace
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: my-namespace:my-sa:d8-rbac-proxy
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: d8:rbac-proxy
subjects:
- kind: ServiceAccount
  name: my-sa
  namespace: my-namespace
```

The Kube-RBAC-Proxy configuration

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-rbac-proxy
data:
  config-file.yaml: |+
    excludePaths:
    - /healthz
  # No authorization for liveness probes is required.
    upstreams:
    - upstream: http://127.0.0.1:8081/
  # The destination address
      path: /
  # The path to the proxy to forward requests to the upstream.
      authorization:
        resourceAttributes:
          namespace: my-namespace
          apiGroup: apps
          apiVersion: v1
          resource: deployments
          subresource: http
          name: my-app
```

According to the configuration, the user must have access to the `my-app` Deployment and its `http` subresource in the `my-namespace` namespace.

Such permissions have the following RBAC form:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kube-rbac-proxy:my-app
  namespace: my-namespace
rules:
- apiGroups: ["apps"]
  resources: ["deployments/http"]
  resourceNames: ["my-app"]
  verbs: ["get", "create", "update", "patch", "delete"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kube-rbac-proxy:my-app
  namespace: my-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kube-rbac-proxy:my-app
subjects:
# All user certificates of Ingress controllers are issued for one specific group.
- kind: Group
  name: ingress-nginx:auth
```

For the Ingress resource, add parameters:

```
nginx.ingress.kubernetes.io/backend-protocol: HTTPS
nginx.ingress.kubernetes.io/configuration-snippet: |
  proxy_ssl_certificate /etc/nginx/ssl/client.crt;
  proxy_ssl_certificate_key /etc/nginx/ssl/client.key;
  proxy_ssl_protocols TLSv1.2;
  proxy_ssl_session_reuse on;
```

Here you can read more about how certificate authentication works.

How do I configure an external load balancer to check if IngressNginxController is available?

In case an Ingress NGINX Controller is deployed behind a load balancer, it is recommended that you configure your load balancer so that it would check the availability of the Ingress NGINX Controller’s endpoints via a health check mechanism, periodically sending either HTTP requests or TCP packets. While it is possible to test the endpoints simply by checking if a relevant TCP port is open, we recommend implementing HTTP checks with the following parameters:

- Protocol: `HTTP`
- Path: `/healthz`
- Port: `80` (or relevant httpPort value in case of using `HostPort` inlet).

How do I configure MetalLB to be accessible from the internal network only?

Below is an example of a MetalLB config with access from the internal network only:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: "nginx"
  inlet: "LoadBalancer"
  loadBalancer:
    sourceRanges:
    - 192.168.0.0/24
```

⚠ The `svcSourceRangeCheck` parameter should be enabled in the `cni-cilium` module for correct work.

How to add extra log fields to a nginx-controller?

Example of adding extra fields:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: "nginx"
  inlet: "LoadBalancer"
  additionalLogFields:
    my-cookie: "$cookie_MY_COOKIE"
```

How to enable HorizontalPodAutoscaling for IngressNginxController?

⚠ HPA mode is possible only for controllers with inlet: `LoadBalancer` or `LoadBalancerWithProxyProtocol`.

HPA mode is possible only for `minReplicas` != `maxReplicas` otherwise deployment `hpa-scaler` will not be created.

HPA is set with attributes `minReplicas` and `maxReplicas` in a IngressNginxController CR.

The IngressNginxController is deployed using DaemonSet. DaemonSet does not provide horizontal scaling capabilities, so `hpa-scaler` Deployment will be created with the HPA resource, which is observing custom metric `prometheus-metrics-adapter-d8-ingress-nginx-cpu-utilization-for-hpa`. If CPU utilization exceeds 50%, the HPA-controller scales `hpa-scaler` Deployment with a new replica (with respect to `minReplicas` and `maxReplicas`).

`hpa-scaler` Deployment has HardPodAntiAffinity, and it will order a new Node (inside its NodeGroup), where one more ingress-controller will be set.

Notes:

- The minimum actual number of IngressNginxController replicas cannot be less than the minimum number of nodes in the node group it is deployed to.
- The maximum actual number of IngressNginxController replicas cannot be greater than the maximum number of nodes in the node group it is deployed to.

How to use IngressClass with IngressClassParameters?

Since version 1.1 IngressNginxController Deckhouse creates an IngressClass object. If you want to use your own IngressClass with your customized IngressClassParameters, you need to add the label `ingress-class.deckhouse.io/external: "true"`:

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    ingress-class.deckhouse.io/external: "true"
  name: my-super-ingress
spec:
  controller: ingress-nginx.deckhouse.io/my-super-ingress
  parameters:
    apiGroup: elbv2.k8s.aws
    kind: IngressClassParams
    name: awesome-class-cfg
```

In this case Deckhouse will not create an IngressClass object and will use your own.

How to disable the collection of detailed Ingress resources statistics?

By default, Deckhouse collects detailed statistics from all Ingress resources in the cluster, which generates a high load on the monitoring system.

To disable statistics collection, add label `ingress.deckhouse.io/discard-metrics: "true"` to the corresponding Namespace or Ingress resource.

Example of disabling statistics (metrics) collection for all Ingress resources in the `review-1` namespace:

```
d8 k label ns review-1 ingress.deckhouse.io/discard-metrics=true
```

Example of disabling statistics (metrics) collection for all `test-site` Ingress resources in the `development` namespace:

```
d8 k label ingress test-site -n development ingress.deckhouse.io/discard-metrics=true
```

How do I correctly drain a node running an IngressNginxController's pods?

There are two ways to gracefully drain a node running IngressNginxController.

1. Using an annotation.

   The annotation will be automatically removed after the operation completes.

   ```
   d8 k annotate node <node_name> update.node.deckhouse.io/draining=user
   ```

2. Using d8 k drain.

   When using the standard d8 k drain command, you must specify the `--force` flag even if `--ignore-daemonsets` is present, since IngressNginxController is deployed using Advanced DaemonSet:

   ```
   d8 k drain <node_name> --delete-emptydir-data --ignore-daemonsets --force
   ```

How to enable Web Application Firewall (WAF)?

Software known as a Web Application Firewall (WAF) is used to protect web applications from Layer 7 attacks. Ingress-nginx controller has a built-in WAF called `ModSecurity' (The Open Worldwide Application Security Project).

ModSecurity is disabled by default.

Enabling ModSecurity

To enable ModSecurity, you must specify the following parameters in the `config` section of the CR IngressNginxController:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: <name_of_the_controller>
spec:
  config:
    enable-modsecurity: "true"
    modsecurity-snippet: |
      Include /etc/nginx/modsecurity/modsecurity.conf
```

After applying the settings, ModSecurity will start working for all traffic passing through this Ingress nginx controller. This uses the audit mode (`DetectionOnly`) and basic recommended configuration.

Setting up ModSecurity

You can configure ModSecurity in two ways:

1. For the entire ingress-nginx controller
   - the necessary directives are described in the section `config.modsecurity-snippet` in CR IngressNginxController, as in the example above.
2. For each CR Ingress separately
   - the necessary directives are described in the annotation `nginx.ingress.kubernetes.io/modsecurity-snippet : |` directly in Ingress manifests.

To enable the execution of rules (and not just logging), add the `SecRuleEngine On` directive as shown below:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: <name_of_the_controller>
spec:
  config:
    enable-modsecurity: "true"
    modsecurity-snippet: |
      Include /etc/nginx/modsecurity/modsecurity.conf
      SecRuleEngine On
```

Adding SecRules

An example of a rule to limit the number of arguments in a request URL. If the number of arguments exceeds 10, the server will reject the request with an error code 400 (Bad Request).

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: <name_of_the_controller>
spec:
  config:
    enable-modsecurity: "true"
    modsecurity-snippet: |
      Include /etc/nginx/modsecurity/modsecurity.conf
      SecRuleEngine On
      SecRule &ARGS "@gt 10" "id:100100,phase:2,deny,status:400,log,auditlog,severity:WARNING,msg:\"too many
args\""
```

A full list and description of the directives can be found in the official documentation.

Enabling OWASP Core Rule Set (CRS)

To enable OWASP Core Rule Set (CRS) add the `config.enable-owasp-modsecurity-crs` in CR IngressNginxController, as in the example below.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: <name_of_the_controller>
spec:
  config:
    enable-modsecurity: "true"
    enable-owasp-modsecurity-crs: "true"
    modsecurity-snippet: |
      Include /etc/nginx/modsecurity/modsecurity.conf
      SecRuleEngine On
```

Examples

> **i** The module lifecycle stage: General Availability

An example for AWS (Network Load Balancer)

When creating a balancer, all zones available in the cluster are used.

In each zone, the balancer receives a public IP. If there is an instance with an Ingress controller in the zone, an A-record with the balancer's IP address from this zone is automatically added to the balancer's domain name.

When there are no instances with an Ingress controller in the zone, then the IP is automatically removed from the DNS.

If there is only one instance with an Ingress controller in a zone, when the pod is restarted, the IP address of the balancer of this zone will be temporarily excluded from DNS.

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
 name: main
spec:
  ingressClass: "nginx"
  inlet: "LoadBalancer"
  loadBalancer:
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
```

An example for GCP / Yandex Cloud / Azure

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
 name: main
spec:
  ingressClass: "nginx"
  inlet: "LoadBalancer"
```

> ⚠ In **GCP**, nodes must have an annotation enabling them to accept connections to external addresses for the NodePort type services.

An example for OpenStack

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main-lbwpp
spec:
  inlet: LoadBalancerWithProxyProtocol
  ingressClass: nginx
  loadBalancerWithProxyProtocol:
    annotations:
      loadbalancer.openstack.org/proxy-protocol: "true"
      loadbalancer.openstack.org/timeout-member-connect: "2000"
```

An example for Bare metal

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: HostWithFailover
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    value: frontend
```

An example for Bare metal (Behind external load balancer, e.g. Cloudflare, Qrator, Nginx+, Citrix ADC, Kemp, etc.)

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: HostPort
  hostPort:
    httpPort: 80
    httpsPort: 443
    behindL7Proxy: true
```

An example for Bare metal (MetalLB BGP LoadBalancer)

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    value: frontend
```

In the case of using MetalLB, its speaker Pods must be run on the same Nodes as the Ingress controller Pods.

The controller must receive real IP addresses of clients — therefore its Service is created with the parameter `externalTrafficPolicy: Local` (disabling cross-node SNAT), and to accept this parameter the MetalLB speaker announce this Service only from those Nodes where the target Pods are running.

For this example, the configuration of the `metallb` module should be as follows:

```
metallb:
 speaker:
   nodeSelector:
     node-role.deckhouse.io/frontend: ""
   tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend
```

An example for Bare metal (MetalLB L2 LoadBalancer)

1. Enable the `metallb` module:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  version: 2
```

2. Create a MetalLoadBalancerClass resource:

> **i** Metallb balancers should be placed on the same nodes as ingress controllers. In typical deployment scenarios, frontend nodes are used for this purpose (to deploy ingress controllers and Metallb load balancers on frontend nodes, use the annotation `node-role.deckhouse.io/frontend: ""` in their manifests).

```
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: ingress
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  isDefault: false
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  type: L2
```

3. Create a IngressNginxController resource:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: main
spec:
  ingressClass: nginx
  inlet: LoadBalancer
  loadBalancer:
    loadBalancerClass: ingress
    annotations:
    # The number of addresses that will be allocated from the pool declared in MetalLoadBalancerClass.
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
  # Label selector and tolerations. Ingress-controller pods must be scheduled on same nodes as MetalLB
speaker pods.
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
  - effect: NoExecute
    key: dedicated.deckhouse.io
    value: frontend
    operator: Equal
```

> **i** When creating an ingress controller, you can also specify certain IP addresses from the pool that will be assigned to it. To specify the addresses that should be assigned to the service, use the annotation `network.deckhouse.io/load-balancer-ips`. If there is more than one desired address, there must also be an annotation `network.deckhouse.io/l2-load-balancer-external-ips-count`, which must specify the number of addresses allocated from the pool (it must not be less than the number of addresses listed in `network.deckhouse.io/load-balancer-ips`). Example of using annotations to assign specific addresses from the pool to the service.

4. The platform will create a service with the type `LoadBalancer`, to which a specified number of addresses will be assigned:

```
d8 k -n d8-ingress-nginx get svc
NAME                  TYPE           CLUSTER-IP      EXTERNAL-IP                                 PORT(S)                      AGE
main-load-balancer    LoadBalancer   10.222.130.11   192.168.2.100,192.168.2.101,192.168.2.102   80:30689/TCP,443:30668/TCP   11s
```

Example of segregating access between public and administrative zones

In many applications, the same backend serves both the public part and the administrative interface. For example:

- `https://example.com` is the public zone;
- `https://admin.example.com` is the administrative zone, access to which must be restricted (`ACL`, `mTLS`, `IP whitelist`, and so on).

For this scenario, we recommend offloading administrative traffic to a separate Ingress controller (with a dedicated Ingress class if necessary) and restricting access to it by using the `spec.acceptRequestsFrom` parameter.

Specifics of using a single Ingress controller

Consider an example where a single Ingress controller is used to serve requests from both the public zone and the administrative interface.

Example of Ingress resource configuration for this case:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: admin-ingress
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: "1.2.3.4/32"
spec:
  ingressClassName: nginx # The Ingress resource for administrative traffic is associated with the same Ingress
controller as the Ingress resource for public traffic.
  rules:
    - host: admin.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: public-ingress
spec:
  ingressClassName: nginx # The Ingress resource for public traffic is associated with the same Ingress
controller as the Ingress resource for administrative traffic.
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
```

With processing and forwarding of X-Forwarded-* headers enabled, the backend can rely on the `x-forwarded-host` header when making authorization decisions. In the example above, it is possible to reach the administrative zone through the Ingress resource that serves public traffic by using `x-forwarded-host`. Therefore, when using this option you must be sure that requests to the Ingress controller come only from trusted sources.

Using separate Ingress controllers

To avoid the situation described above (when, with processing and forwarding of X-Forwarded-* headers enabled it is possible, for example, to reach the administrative zone via the Ingress resource that serves public traffic by using `x-forwarded-host`), we recommend that you:

- configure access rules at the Ingress resource level,
- use separate Ingress controllers,
- restrict which source addresses are allowed to connect to the Ingress controllers.

Example of Ingress resource configuration for this case:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: admin-ingress
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: "1.2.3.4/32"
spec:
  ingressClassName: admin-nginx # The Ingress resource for administrative traffic is associated with a separate
Ingress controller.
  rules:
    - host: admin.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: public-ingress
spec:
  ingressClassName: public-nginx # The Ingress resource for public traffic is associated with a separate Ingress
controller.
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
```

Example of an Ingress controller that serves administrative Ingress resources and accepts connections only from specified subnets:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: admin
spec:
  ingressClass: admin-nginx
  inlet: HostPort
  acceptRequestsFrom:
    - 1.2.3.4/32
    - 10.0.0.0/16
  hostPort:
    httpPort: 80
    httpsPort: 443
    behindL7Proxy: true
```

In this example:

- The Ingress controller is exposed on node ports through the `HostPort` inlet.
- The `acceptRequestsFrom` parameter allows connections to the controller only from the listed subnets.
- Even if an external load balancer or client can set its own `X-Forwarded-*` header values, the decision whether to allow the connection to reach the controller is made based on the actual source address, not on headers.
- Administrative Ingress resources (in this example `admin-ingress`) are served by this controller according to the configured Ingress class.

Example of an Ingress controller that serves Ingress resources for public traffic:

```
apiVersion: deckhouse.io/v1
kind: IngressNginxController
metadata:
  name: public
spec:
  ingressClass: public-nginx
  inlet: HostPort
  hostPort:
    httpPort: 8080
    httpsPort: 8443
    behindL7Proxy: true
```

The istio module

i Available with limitations in: **CE, BE, SE, SE+, CSE Lite (1.73)**

Available without limitations in: **EE**

i The module lifecycle stage: General Availability

i Web interfaces associated with the module: istio

Compatibility table for supported versions

Istio version	K8S versions supported by Istio	Status in D8
1.25	1.29, 1.30, 1.31, 1.32, 1.33, 1.34, 1.35	Supported
1.21	1.26, 1.27, 1.28, 1.29, 1.30, 1.31, 1.32, 1.33, 1.34, 1.35	Deprecated and will be deleted

What issues does Istio help to resolve?

Istio is a framework for managing network traffic on a centralized basis that implements the Service Mesh approach.

Istio solves the tasks for applications:

- Using Mutual TLS.
- Access authorization between services.
- Request routing.
- Managing request balancing between service Endpoints.
- Improving Observability.
- Organizing a multi-datacenter cluster by joining clusters into a single Service Mesh (multicluster).
- Grouping isolated clusters into a federation with the ability to provide native (in the Service Mesh sense) access to selected services .

Mutual TLS

Mutual TLS is the main method of mutual service authentication. It is based on the fact that all outgoing requests are verified using the server certificate, and all incoming requests are verified using the client certificate. After the verification is complete, the sidecar-proxy can identify the remote node and use these data for authorization or auxiliary purposes.

Each service gets its own identifier of the following format: `<TrustDomain>/ns/<Namespace>/sa/<ServiceAccount>` where `TrustDomain` is the cluster domain in our case. You can assign your own ServiceAccount to each service or use the regular “default” one. The service ID can be used for authorization and other purposes. This is the identifier used as a name to validate against in TLS certificates.

You can redefine this settings at the Namespace level.

Authorization

The AuthorizationPolicy resource is responsible for managing authorization. Once this resource is created for the service, the following algorithm is used for determining the fate of the request:

- The request is denied if it falls under the DENY policy.
- The request is allowed if there are no ALLOW policies for the service.
- The request is allowed if it falls under the ALLOW policy.
- In all other cases, the request is denied.

In other words, if you explicitly deny something, then only this restrictive rule will work. On the other hand, if you explicitly allow something, only explicitly authorized requests would be allowed (however, restrictions will have precedence).

You can use the following arguments for defining authorization rules:

- Service IDs and wildcard expressions based on them (`mycluster.local/ns/myns/sa/myapp` or `mycluster.local/*`)
- Namespace
- IP ranges
- HTTP headers
- JWT tokens

Request routing

VirtualService is the main resource for routing control; it allows you to override the destination of an HTTP or TCP request. Routing decisions can be based on the following parameters:

- Host or other headers
- URI
- Method (GET, POST, etc.)
- Pod labels or the namespace of the request source
- dst-IP or dst-port for non-HTTP requests

Managing request balancing between service Endpoints

DestinationRule is the main resource for managing request balancing; it allows you to configure the details of requests leaving the Pods:

- Limits/timeouts for TCP
- Balancing algorithms between Endpoints
- Rules for detecting problems on the Endpoint side to take it out of balancing
- Encryption details

⚠ All customizable limits apply to each client Pod individually (on a per Pod basis)! Suppose you limited a service to one TCP connection. In this case, if you have three client Pods, the service will get three incoming connections.

Observability

Tracing

Istio makes it possible to collect application traces and inject trace headers if there are none. In doing so, however, you have to keep in mind the following:

- If a request initiates secondary requests for a service, they must inherit the trace headers by means of the application.
- You will need to install Jaeger to collect and display traces.

Grafana

The standard module bundle includes the following additional dashboards:

- Dashboard for evaluating the throughput and success of requests/responses between applications.
- Dashboard for evaluating control plane performance and load.

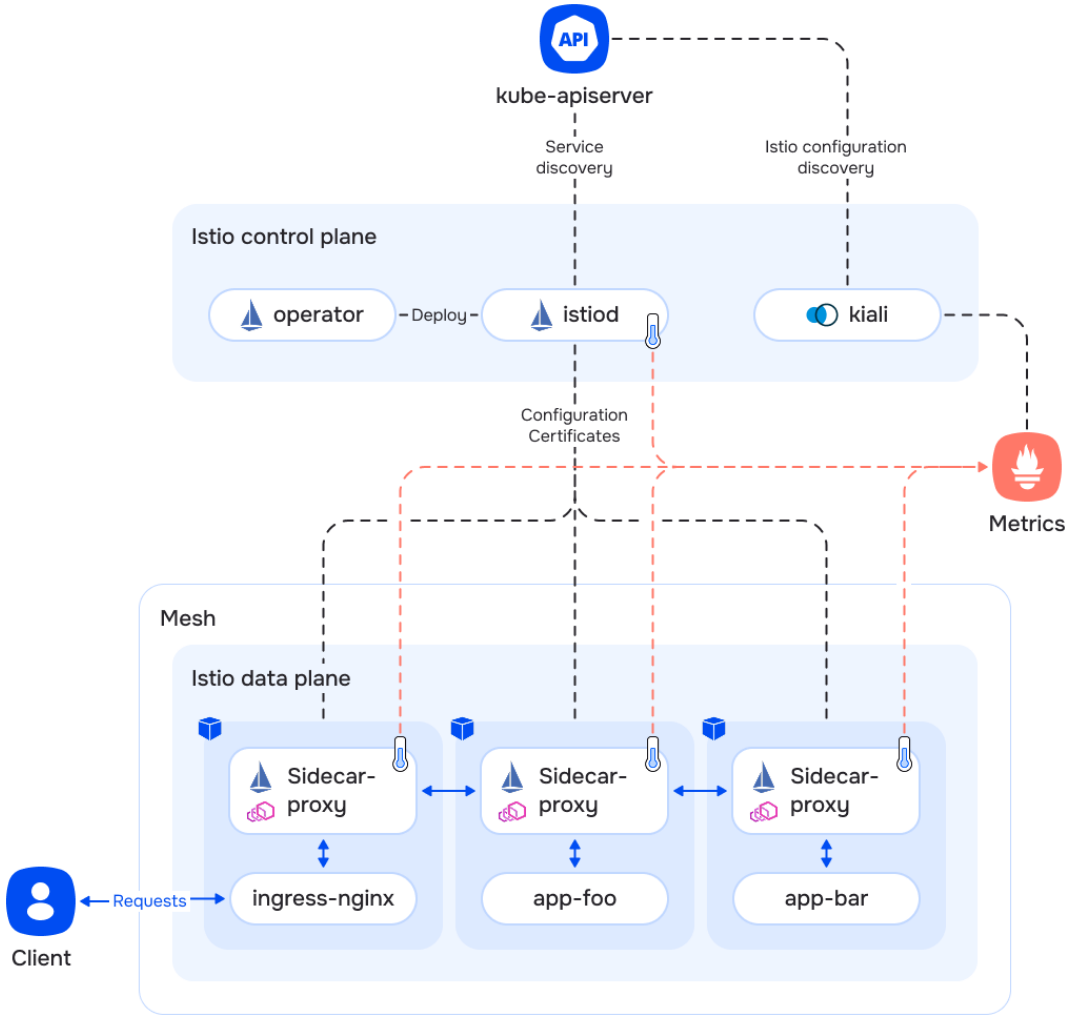
Kiali

Kiali is a tool for visualizing your application’s service tree. It allows you to quickly assess the situation in the network connectivity by visualizing the requests and their quantitative characteristics directly on the scheme.

Architecture of the cluster with Istio enabled

The cluster components are divided into two categories:

- Control plane — managing and maintaining services; “control-plane” usually refers to istiod Pods;
- Data plane — mediating and controlling all network communication between microservices, it is composed of a set of sidecar-proxy containers.



All data plane services are grouped into a mesh with the following features:

- It has a common namespace for generating service ID in the form `<TrustDomain>/ns/<Namespace>/sa/<ServiceAccount>`. Each mesh has a TrustDomain ID (in our case, it is the same as the cluster domain), e.g. `mycluster.local/ns/myns/sa/myapp`.
- Services within a single mesh can authenticate each other using trusted root certificates.

Control plane components:

- `istiod` — the main service with the following tasks:
  - Continuous connection to the Kubernetes API and collecting information about services.
  - Processing and validating all Istio-related Custom Resources using the Kubernetes Validating Webhook mechanism.
  - Configuring each sidecar proxy individually:

        - Generating authorization, routing, balancing rules, etc..
        - Distributing information about other application services in the cluster.
        - Issuing individual client certificates for implementing Mutual TLS. These certificates are unrelated to the certificates that Kubernetes uses for its own service needs.
    - Automatic tuning of manifests that describe application Pods via the Kubernetes Mutating Webhook mechanism:
        - Injecting an additional sidecar-proxy service container.
        - Injecting an additional init container for configuring the network subsystem (configuring DNAT to intercept application traffic).
        - Routing readiness and liveness probes through the sidecar-proxy.
- `operator` — installs all the resources required to operate a specific version of the control plane.
- `kiali` — dashboard for monitoring and controlling Istio resources as well as user services managed by Istio that allows you:
    - Visualize inter-service connections.
    - Diagnose problem inter-service connections.
    - Diagnose the control plane state.

The Ingress controller must be refined to receive user traffic:

- You need to add sidecar-proxy to the controller Pods. It only handles traffic from the controller to the application services (the `enableIstioSidecar` parameter of the `IngressNginxController` resource).
- Services not managed by Istio continue to function as before, requests to them are not intercepted by the controller sidecar.
- Requests to services running under Istio are intercepted by the sidecar and processed according to Istio rules (read more about activating Istio to work with the application).

The istiod controller and sidecar-proxy containers export their own metrics that the cluster-wide Prometheus collects.

Application service architecture with Istio enabled

Details

- Each service Pod gets a sidecar container — sidecar-proxy. From the technical standpoint, this container contains two applications:
    - **Envoy** proxies service traffic. It is responsible for implementing all the Istio functionality, including routing, authentication, authorization, etc.
    - **pilot-agent** is a part of Istio. It keeps the Envoy configurations up to date and has a built-in caching DNS server.
- Each Pod has a DNAT configured for incoming and outgoing service requests to the sidecar-proxy. The additional init container is used for that. Thus, the traffic is routed transparently for applications.
- Since incoming service traffic is redirected to the sidecar-proxy, this also applies to the readiness/liveness traffic. The Kubernetes subsystem that does this doesn't know how to probe containers under Mutual TLS. Thus, all the existing probes are automatically reconfigured to use a dedicated sidecar-proxy port that routes traffic to the application unchanged.
- You have to configure the Ingress controller to receive requests from outside the cluster:
    - The controller's Pods have additional sidecar-proxy containers.
    - Unlike application Pods, the Ingress controller's sidecar-proxy intercepts only outgoing traffic from the controller to the services. The incoming traffic from the users is handled directly by the controller itself;
- Ingress resources require refinement in the form of adding annotations:
    - `nginx.ingress.kubernetes.io/service-upstream: "true"` — the Ingress controller will use the service's ClusterIP as upstream instead of the Pod addresses. In this case, traffic balancing between the Pods is handled by the sidecar-proxy. Use this option only if your service has a ClusterIP.
    - `nginx.ingress.kubernetes.io/upstream-vhost: "myservice.myns.svc"` — the Ingress controller's sidecar-proxy makes routing decisions based on the Host header. If this annotation is omitted, the controller will leave a header with the site address (e.g. `Host: example.com`).
- Resources of the Service type do not require any adaptation and continue to function properly. Just like before, applications have access to service addresses like `servicename`, `servicename.myns.svc`, etc;
- DNS requests from within the Pods are transparently redirected to the sidecar-proxy for processing:
    - This way, domain names of the services in the neighboring clusters can be disassociated from their addresses.

User request lifecycle

Application with Istio turned off

Application with Istio turned on

Activating Istio to work with the application

The main purpose of the activation is to add a sidecar container to the application Pods so that Istio can manage the traffic.

The sidecar-injector is a recommended way to add sidecars. Istio can inject sidecar containers into user Pods using the Admission Webhook mechanism. You can configure it using labels and annotations:

- A label attached to a namespace allows the sidecar-injector to identify a group of Pods to inject sidecar containers into:
    - `istio-injection=enabled` — use the global version of Istio (`spec.settings.globalVersion` in `ModuleConfig`);
    - `istio.io/rev=v1x21` — use the specific Istio version for a given namespace;
    - `istio.io/rev=default` — use the global version of Istio (`spec.settings.globalVersion` in `ModuleConfig`).
- The `sidecar.istio.io/inject` (`"true"` or `"false"`) **Pod** annotation lets you redefine the `sidecarInjectorPolicy` policy locally. These annotations work only in namespaces to which the above labels are attached.

It is also possible to add the sidecar to an individual pod in namespace without the `istio-injection=enabled` or `istio.io/rev=vXxYZ` labels by setting the `sidecar.istio.io/inject=true` Pod label.

⚠ Istio-proxy, running as a sidecar container, consumes resources and adds overhead.

Overhead added by Istio-proxy:

- Any incoming DNAT request is forcibly intercepted by the Envoy proxy. Envoy analyzes the request and forwards it by establishing a new connection. On the receiver side, the process is identical: traffic first reaches the "receiving" Envoy and only then is passed to the application itself.
- Each Envoy instance stores information about all services in the cluster; therefore, scaling the cluster leads to a linear increase in Envoy's RAM consumption due to storing the full service map. Using the Sidecar CustomResource, the configuration is filtered, and only the necessary minimum of data is delivered to Envoy.

⚠ The EnvoyFilter interface can be controlled by Lua plugins, but it is an internal control mechanism for implementing the Istio functionality. It must not be used in a user configuration, as doing so would compromise the integrity of the system.

It is also important to get the Ingress controller and the application's Ingress resources ready:

- Enable `enableIstioSidecar` of the `IngressNginxController` resource.
- Add annotations to the application's Ingress resources:
  - `nginx.ingress.kubernetes.io/service-upstream: "true"` — the Ingress controller will use the service's ClusterIP as upstream instead of the Pod addresses. In this case, traffic balancing between the Pods is now handled by the sidecar-proxy. Use this option only if your service has a ClusterIP.
  - `nginx.ingress.kubernetes.io/upstream-vhost: "myservice.myns.svc"` — the Ingress controller's sidecar-proxy makes routing decisions based on the Host header. If this annotation is omitted, the controller will leave a header with the site address (e.g. `Host: example.com`).

Federation and multicluster

Deckhouse supports two schemes of inter-cluster interaction:

- federation
- multicluster

Below are their fundamental differences:

- The federation aggregates multiple sovereign (independent) clusters:
  - Each cluster has its own namespace (for Namespace, Service, etc.);
  - Access to individual services between clusters is clearly defined.
- The multicluster aggregates co-dependent clusters:
  - Cluster namespaces are shared — each service is available to neighboring clusters as if it were running in a local cluster (unless authorization rules prohibit that).

Federation

Requirements for clusters

- Each cluster must have a unique domain in the `clusterDomain` parameter of the resource ClusterConfiguration. The default value is `cluster.local`.
- Pod and Service subnets in the `podSubnetCIDR` and `serviceSubnetCIDR` parameters of the resource ClusterConfiguration can be the same.
- Each cluster must have a unique domain in the `clusterDomain` parameter of the resource ClusterConfiguration. Please note that none of the clusters should use the domain `cluster.local`, which is the default setting.

  > **i** `cluster.local` is an unmodified alias for the local cluster domain. When specifying `cluster.local` as a principals in the AuthorizationPolicy, it will always refer to the local cluster, even if there is another cluster in the mesh with `clusterDomain` explicitly defined as `cluster.local`. source

- There are no requirements for unique subnets of modules and services in the `podSubnetCIDR` and `serviceSubnetCIDR` parameters of the resource ClusterConfiguration when clusters are operating in a federation.
  - When analyzing HTTP and HTTPS traffic *(in Istio terminology)*, you can identify them and decide on further routing or blocking based on their headers.
  - At the same time, when analyzing TCP traffic *(in Istio terminology)*, it is possible to identify them and decide on further routing or blocking based only on their destination IP address or port number.

    > **i** Istio operates in the multi-network mode — pods from different clusters can only communicate with each other through the Istio ingress gateway. Direct communication between pods of different clusters is not supported.

General principles of federation

- Federation requires mutual trust between clusters. Thereby, to use federation, you have to make sure that both clusters (say, A and B) trust each other. This is achieved by a mutual exchange of root certificates.
- You also need to share information about government services to use the federation. You can do that using ServiceEntry resource. A service entry defines the public ingress-gateway address of the B cluster so that services of the A cluster can communicate with the bar service in the B cluster.

Enabling the federation

Enabling federation (via the `istio.federation.enabled = true` module parameter) results in the following activities:

- The `ingressgateway` service is added to the cluster. Its task is to proxy mTLS traffic coming from outside of the cluster to application services.
- A service gets added to the cluster that exports the following cluster metadata to the outside:
  - Istio root certificate (accessible without authentication).
  - List of public services in the cluster (available only for authenticated requests from neighboring clusters).
  - List of public addresses of the `ingressgateway` service (available only for authenticated requests from neighboring clusters).

Managing the federation

To establish a federation, you must:

- Create a set of `IstioFederation` resources in each cluster that describe all the other clusters.
  - After successful auto-negotiation between clusters, the status of `IstioFederation` resource will be filled with neighbour's public and private metadata (`status.metadataCache.public` and `status.metadataCache.private`).
- Add the `federation.istio.deckhouse.io/public-service: ""` label to each resource(`service`) that is considered public within the federation.
  - In the other federation clusters, a corresponding `ServiceEntry` will be created for each `service`, leading to the `ingressgateway` of the original cluster.

> ⚠ In the `.spec.ports` section of `services`, each port must have the `name` field filled.

Multicluster

Requirements for clusters

- Cluster domains in the `clusterDomain` parameter of the resource ClusterConfiguration must be the same for all multicluster members. The default value is `cluster.local`.
- Pod and Service subnets in the `podSubnetCIDR` and `serviceSubnetCIDR` parameters of the resource ClusterConfiguration must be unique for each multicluster member.
  - When analyzing HTTP and HTTPS traffic *(in Istio terminology)*, you can identify them and decide on further routing or blocking based on their headers.
  - At the same time, when analyzing TCP traffic *(in Istio terminology)*, it is possible to identify them and decide on further routing or blocking based only on their destination IP address or port number.

  > **i**
  >
  > If the IP addresses of services or pods in different clusters match, requests from other pods in other clusters may mistakenly fall under the Istio's rules. The intersection of subnets of services and pods is not recommended (source).
  >
  > Istio operates in the multi-network mode — pods from different clusters can only communicate with each other through the Istio ingress gateway. Direct communication between pods of different clusters is not supported.

General principles

- Multicluster requires mutual trust between clusters. Thereby, to use multiclustering, you have to make sure that both clusters (say, A and B) trust each other. From a technical point of view, this is achieved by a mutual exchange of root certificates.
- Istio connects directly to the API server of the neighboring cluster to gather information about its services. This Deckhouse module takes care of the corresponding communication channel.

Enabling the multicluster

Enabling the multicluster (via the `istio.multicluster.enabled = true` module parameter) results in the following activities:

- A proxy is added to the cluster to publish access to the API server via the standard Ingress resource:
  - Access through this public address is secured by authorization based on Bearer tokens signed with trusted keys. Deckhouse automatically exchanges trusted public keys during the mutual configuration of the multicluster.
  - The proxy itself has read-only access to a limited set of resources.
- A service gets added to the cluster that exports the following cluster metadata to the outside:
  - Istio root certificate (accessible without authentication).
  - The public API server address (available only for authenticated requests from neighboring clusters).
  - List of public addresses of the `ingressgateway` service (available only for authenticated requests from neighboring clusters).
  - Server public keys to authenticate requests to API server and to private metadata (see above).

Managing the multicluster

To create a multicluster, you need to create a set of `IstioMulticluster` resources in each cluster that describe all the other clusters.

In case of issues when working with a multi-cluster, it is necessary to check in each cluster:

1. The status of the `IstioMultiCluster` resources. To do this, run the command `d8 k describe istiomulticluster cluster-name`. It is important that the resource status shows `Root CA` and that the `Public Last Fetch Timestamp` field has a recent timestamp.
2. The `Ingress Gateways` field of the `IstioMultiCluster` resource should contain the IP address of the second cluster's `IngressGateway`.
3. Using the `istioctl` utility (how to install...):

```
istioctl remote-clusters -i d8-istio
NAME          SECRET                                    STATUS     ISTIOD
cluster-b     d8-istio/istio-remote-secret-cluster-b     synced     istiod-v1x21-5c57d85b54-k8pl7
```

Estimating overhead

Using Istio will incur additional resource costs for both **control-plane** (istiod controller) and **data-plane** (istio-sidecars).

control-plane

The istiod controller continuously monitors the cluster configuration, compiles the settings for the istio-sidecars and distributes them over the network. Accordingly, the more applications and their instances, the more services, and the more frequently this configuration changes, the more computational resources are required and the greater the load on the network. Two approaches are supported to reduce the load on controller instances:

- horizontal scaling (module configuration `controlPlane.replicasManagement`) — the more controller instances, the fewer instances of istio-sidecars to serve for each controller and the less CPU and network load.
- data-plane segmentation using the *Sidecar* resource (recommended approach) — the smaller the scope of an individual istio-sidecar, the less data in the data-plane needs to be updated and the less CPU and network overhead.

A rough estimate of overhead for a control-plane instance that serves 1000 services and 2000 istio-sidecars is 1 vCPU and 1.5 GB RAM.

data-plane

The consumption of data-plane resources (istio-sidecar) is affected by many factors:

- number of connections,
- the intensity of requests,
- size of requests and responses,
- protocol (HTTP/TCP),
- number of CPU cores,
- complexity of Service Mesh configuration.

A rough estimate of the overhead for an istio-sidecar instance is 0.5 vCPU for 1000 requests/sec and 50 MB RAM. istio-sidecars also increase latency in network requests — about 2.5ms per request.

Third-party components

List of third-party software used in the `istio` module:

- Istio 1.21.6

  License: Apache License 2.0

  An open platform to connect, manage, and secure microservices.

- Istio 1.25.2

  License: Apache License 2.0

  An open platform to connect, manage, and secure microservices.

- Kiali 1.81.0

  License: Apache License 2.0

  Visualisation tool for the istio service mesh topology, and features like circuit breakers or request rates.

- Kiali 2.7.1

  License: Apache License 2.0

  Visualisation tool for the istio service mesh topology, and features like circuit breakers or request rates.

Configuration

> i Available with limitations in: **CE, BE, SE, SE+, CSE Lite (1.73)**
>
> Available without limitations in: **EE**

> i The module lifecycle stage: General Availability

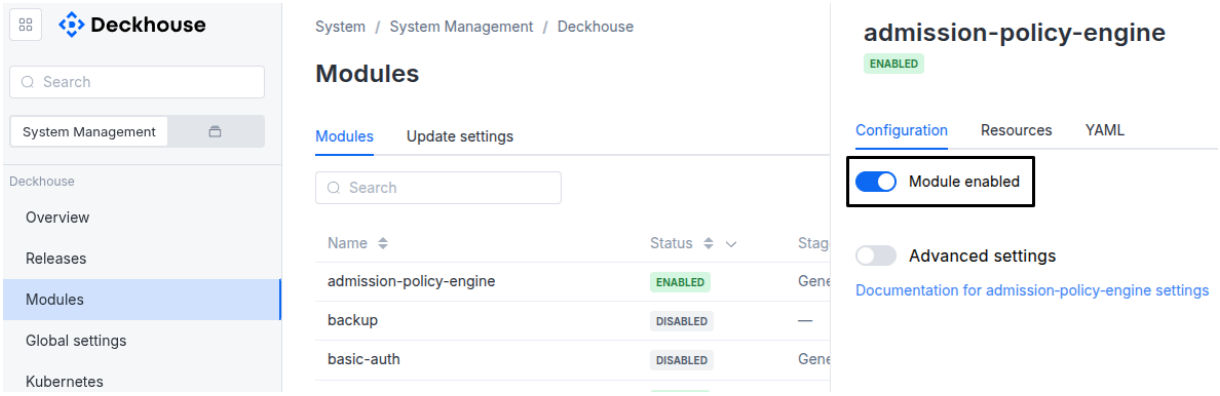
The module has 17 alerts.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `istio` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable istio
```

- **Using ModuleConfig `istio`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `istio` (create it if necessary);

  Example of a manifest to enable module `istio`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: istio
spec:
  enabled: true
```

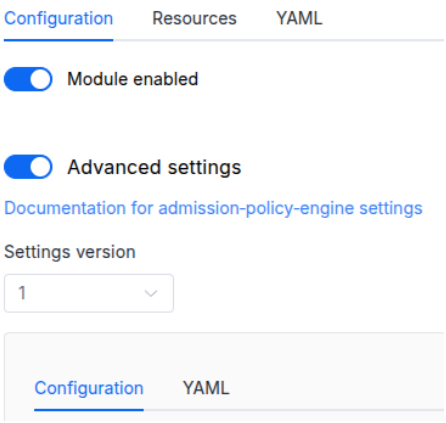
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `istio` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

Example:



You can also edit the ModuleConfig object `istio` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `istio`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `istio` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc istio
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `istio` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `istio`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: istio
  spec:
    version: 3
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Delete `.settings.auth.password`.

- Updates from version **2** to **3**:
  - Move `.settings.enableHTTP10` to `.settings.dataPlane.enableHTTP10`.
  - Move `.settings.proxyConfig` to `.settings.dataPlane.proxyConfig`.

Parameters

Schema version: 3

- **settings**
  object
  - **settings.additionalVersions**
    array of strings

    Additional versions of Istio control plane to install. You can use specific namespace labels (`istio.io/rev=`) to switch between installed revisions.

    Default: `[]`

    - **Element of the array**
      string

      Pattern: `^[0-9]+\.[0-9]+$`

  - **settings.alliance**
    object

Available in editions: EE

Common options both for federation and multicluster.

- **settings.alliance.ingressGateway**
  object

  Available in editions: EE

  ingressgateway settings.

  - **settings.alliance.ingressGateway.advertise**
    array of objects

    Available in editions: EE

    The actual addresses that will be announced to remote clusters for organizing intercluster application requests. If not specified, the addresses will be discovered automatically.

    Default: `[]`

    Examples:

    ```
    advertise:
    - address: 172.16.0.5
      port: 15443
    ```

    ```
    advertise:
    - address: somehost.example.com
      port: 15443
    ```

    - **settings.alliance.ingressGateway.advertise.address**
      string

      Required value

      Pattern: `^([a-z0-9]([a-z0-9-]{0,61}[a-z0-9])?\.)+[a-z]{2,}$`

      Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

      Examples:

      ```
      address: somehost.example.com
      ```

      ```
      address: 172.16.0.5
      ```

    - **settings.alliance.ingressGateway.advertise.port**
      integer

      Required value

      Allowed values: `1024 <= X <= 65535`

  - **settings.alliance.ingressGateway.gatewayPodAnnotations**
    object

    Available in editions: EE

    Additional annotations for ingressgateway DaemonSet pods.

    Example:

    ```
    gatewayPodAnnotations:
      checksum/config: xyz-123
    ```

  - **settings.alliance.ingressGateway.inlet**
    string

    Available in editions: EE

    The method for exposing ingressgateway.

    - `LoadBalancer` — is a recommended method if you have a cloud-based cluster and it supports Load Balancing.
    - `NodePort` — for installations that do not have the LB.

    Default: `LoadBalancer`

    Allowed values: `LoadBalancer`, `NodePort`

    Example:

    ```
    inlet: LoadBalancer
    ```

  - **settings.alliance.ingressGateway.nodePort**
    object

    Available in editions: EE

    Special settings for NodePort inlet.

    Examples:

    ```
    nodePort: {}
    ```

```
nodePort:
  port: 30001
```

- **settings.alliance.ingressGateway.nodePort.port**
  integer

  Static port number for NodePort-type Service. Must be in range, set by kube-apiserver –service-node-port-range argument (default is 30000-32767).

  Allowed values: `1024 <= X <= 65535`

- **settings.alliance.ingressGateway.nodeSelector**
  object

  Available in editions: EE

  ingressgateway DaemonSet nodeSelector.

  The same as the `spec.nodeSelector` pod parameter in Kubernetes.

  Example:

  ```
  nodeSelector:
    type: ingress
  ```

- **settings.alliance.ingressGateway.serviceAnnotations**
  object

  Available in editions: EE

  Additional service annotations. They can be used, e.g., for configuring a local LB in the Yandex Cloud (using the `yandex.cpi.flant.com/listener-subnet-id` annotation).

  Example:

  ```
  serviceAnnotations:
    yandex.cpi.flant.com/listener-subnet-id: xyz-123
  ```

- **settings.alliance.ingressGateway.tolerations**
  array of objects

  Available in editions: EE

  ingressgateway DaemonSet tolerations.

  The same as `spec.tolerations` for the Kubernetes pod.

  Example:

  ```
  tolerations:
  - operator: Exists
  ```

  - **settings.alliance.ingressGateway.tolerations.effect**
    string
  - **settings.alliance.ingressGateway.tolerations.key**
    string
  - **settings.alliance.ingressGateway.tolerations.operator**
    string
  - **settings.alliance.ingressGateway.tolerations.tolerationSeconds**
    integer
  - **settings.alliance.ingressGateway.tolerations.value**
    string

- **settings.auth**
  object

  Options related to authentication or authorization in the application.

  Example:

  ```
  auth:
    externalAuthentication:
      authURL: https://dex.d8.svc.cluster.local/dex/auth
      authSignInURL: https://example.com/dex/sign_in
    allowedUserGroups:
    - admins
  ```

  - **settings.auth.allowedUserEmails**
    array of strings

    An array of emails of users that can access module’s public web interfaces.

    This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

  - **settings.auth.allowedUserGroups**
    array of strings

    An array of user groups that can access module’s public web interfaces.

    This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

    Caution! Note that you must add those groups to the appropriate field in the DexProvider config if this module is used together with the user-authn one.

  - **settings.auth.externalAuthentication**
    object

Parameters to enable external authentication based on the Ingress NGINX external-auth mechanism that uses the Nginx auth_request module.

> **i** External authentication is enabled automatically if the user-authn module is enabled.

- **settings.auth.externalAuthentication.authSignInURL**
  string

  The URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response code).

  Example:

  ```
  authSignInURL: https://example.com/dex/sign_in
  ```

- **settings.auth.externalAuthentication.authURL**
  string

  The URL of the authentication service.

  If the user is authenticated, the service should return an HTTP 200 response code.

  Example:

  ```
  authURL: https://example.com/dex/auth
  ```

- **settings.auth.satisfyAny**
  boolean

  Enables single authentication.

  If used together with the whitelistSourceRanges parameter, it authorizes all the users from above networks (no need to enter a username and password).

  Default: `false`

  Example:

  ```
  satisfyAny: true
  ```

- **settings.auth.whitelistSourceRanges**
  array of strings

  An array if CIDRs that are allowed to authenticate in module's public web interfaces.

  Example:

  ```
  whitelistSourceRanges:
  - 1.1.1.1/32
  ```

- **settings.ca**
  object

  Explicitly specified root certificate. It signs individual service certificates to use in mutual TLS connections. To create a certificate, you can use the example, in which `basicConstraints = CA:FALSE` needs to be replaced with `basicConstraints = CA:TRUE`.

  - **settings.ca.cert**
    string

    The root or intermediate certificate in PEM format.

  - **settings.ca.chain**
    string

    A certificate chain in PEM format if `cert` is an intermediate certificate.

  - **settings.ca.key**
    string

    The key to the root certificate in PEM format.

  - **settings.ca.root**
    string

    The root certificate in PEM format if `cert` is an intermediate certificate.

- **settings.controlPlane**
  object

  istiod specific settings.

  - **settings.controlPlane.nodeSelector**
    object

    Optional `nodeSelector` for istiod. The same as the `spec.nodeSelector` pod parameter in Kubernetes.

    If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.controlPlane.replicasManagement**
    object

    Replication management settings and scaling of istiod.

    Examples:

```
replicasManagement:
  mode: Standard
```

```
replicasManagement:
  mode: Static
  static:
    replicas: 3
```

```
replicasManagement:
  mode: HPA
  hpa:
    minReplicas: 2
    maxReplicas: 5
    metrics:
    - type: CPU
      targetAverageUtilization: 80
```

- **settings.controlPlane.replicasManagement.hpa**
  object

  Options for replicas management for the `HPA` mode.

  - **settings.controlPlane.replicasManagement.hpa.maxReplicas**
    number

    Required value

    The upper limit for the number of replicas to which the HPA can scale up. It cannot be less that `minReplicas`.

    Allowed values: `1 <= X`

  - **settings.controlPlane.replicasManagement.hpa.metrics**
    array of objects

    Required value

    The HPA will use these metrics to decide whether to increase or decrease the number of replicates.

    - **settings.controlPlane.replicasManagement.hpa.metrics.targetAverageUtilization**
      number

      Required value

      The target value of the average of the resource metric across all relevant pods, represented as a percentage of the requested value of the resource for the pods.

      Allowed values: `1 <= X <= 100`

    - **settings.controlPlane.replicasManagement.hpa.metrics.type**
      string

      Required value

      Metric type.

      Allowed values: `CPU`

  - **settings.controlPlane.replicasManagement.hpa.minReplicas**
    number

    Required value

    The lower limit for the number of replicas to which the HPA can scale down.

    Allowed values: `1 <= X`

- **settings.controlPlane.replicasManagement.mode**
  string

  Replicas management mode:

  - `Standard` — replicas management and scaling mode according to the global fault tolerance mode (the highAvailability parameter);
  - `Static` — the mode, where the number of replicas is specified explicitly (the static.replicas parameter);
  - `HPA` — the mode, where the number of replicas is calculated automatically using HPA based on CPU usage. You can configure this mode by modifying parameters in the hpa parameter section.

  Default: `Standard`

  Allowed values: `Standard`, `Static`, `HPA`

- **settings.controlPlane.replicasManagement.static**
  object

  Options for replicas management for the `Static` mode.

  - **settings.controlPlane.replicasManagement.static.replicas**
    number

    Required value

    Desired number of replicas.

    Allowed values: `1 <= X`

- **settings.controlPlane.resourcesManagement**
  object

Settings for CPU and memory requests and limits by istiod pods.

Examples:

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: InPlaceOrRecreate
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

```
resourcesManagement:
  mode: Static
  static:
    requests:
      cpu: 55m
      memory: 256Mi
    limits:
      cpu: '2'
      memory: 2Gi
```

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: Auto
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

- **settings.controlPlane.resourcesManagement.mode**
  string

  Resource management mode:

  - `Static` — allows you to specify requests/limits. The parameters of this mode are defined in the `static` parameter section;
  - `VPA` — uses VPA. You can configure this mode by modifying parameters in the `vpa` parameter section.

  Default: `VPA`

  Allowed values: `VPA`, `Static`

- **settings.controlPlane.resourcesManagement.static**
  object

  Resource management options for the `Static` mode.

  - **settings.controlPlane.resourcesManagement.static.limits**
    object

    Configuring CPU and memory limits.

    - **settings.controlPlane.resourcesManagement.static.limits.cpu**
      string

      Configuring CPU limits.

      Pattern: `^[0-9]+m?$`

    - **settings.controlPlane.resourcesManagement.static.limits.memory**
      string or number

      Configuring memory limits.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.controlPlane.resourcesManagement.static.requests**
    object

    Resource requests settings for pods.

    - **settings.controlPlane.resourcesManagement.static.requests.cpu**
      string

      Configuring CPU requests.

      Pattern: `^[0-9]+m?$`

    - **settings.controlPlane.resourcesManagement.static.requests.memory**
      string or number

      Configuring memory requests.

Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.controlPlane.resourcesManagement.vpa**
  object

  Resource management options for the `VPA` mode.

  - **settings.controlPlane.resourcesManagement.vpa.cpu**
    object

    CPU-related VPA settings.

    - **settings.controlPlane.resourcesManagement.vpa.cpu.limitRatio**
      number

      The CPU limits/requests ratio.

      This ratio is used for calculating the initial CPU limits for a pod.

      If this parameter is set, the VPA will recalculate the CPU limits while maintaining the specified limits/requests ratio.

    - **settings.controlPlane.resourcesManagement.vpa.cpu.max**
      string or number

      The maximum value that the VPA can set for the CPU requests.

      Default: `2`

      Pattern: `^[0-9]+m?$`

    - **settings.controlPlane.resourcesManagement.vpa.cpu.min**
      string or number

      The minimum value that the VPA can set for the CPU requests.

      Default: `50m`

      Pattern: `^[0-9]+m?$`

  - **settings.controlPlane.resourcesManagement.vpa.memory**
    object

    Memory-related VPA settings.

    - **settings.controlPlane.resourcesManagement.vpa.memory.limitRatio**
      number

      The memory limits/requests ratio.

      This ratio is used for calculating the initial memory limits for a pod.

      If this parameter is set, the VPA will recalculate the memory limits while maintaining the specified limits/requests ratio.

    - **settings.controlPlane.resourcesManagement.vpa.memory.max**
      string or number

      The maximum memory requests the VPA can set.

      Default: `2Gi`

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    - **settings.controlPlane.resourcesManagement.vpa.memory.min**
      string or number

      The minimum memory requests the VPA can set.

      Default: `256Mi`

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.controlPlane.resourcesManagement.vpa.mode**
    string

    VPA operating mode.

    - `Initial` — VPA sets initial values for pod resource requests and limits at pod creation time. Resource values are not changed automatically afterwards.
    - `InPlaceOrRecreate` — VPA attempts to update pod resources in place when supported by the cluster. If in-place updates are not possible, the pod will be recreated.
    - `Auto` — VPA automatically recreates pods to apply updated resource values.

      > **i** Starting from Deckhouse version 1.75, this mode is considered legacy. It is recommended to use `InPlaceOrRecreate` instead.

    Default: `InPlaceOrRecreate`

    Allowed values: `Initial`, `InPlaceOrRecreate`, `Auto`

- **settings.controlPlane.tolerations**
  array of objects

  Optional `tolerations` for istiod. The same as `spec.tolerations` for the Kubernetes pod.

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.controlPlane.tolerations.effect**
    string

        - **settings.controlPlane.tolerations.key**
          string
        - **settings.controlPlane.tolerations.operator**
          string
        - **settings.controlPlane.tolerations.tolerationSeconds**
          integer
        - **settings.controlPlane.tolerations.value**
          string
  - **settings.dataPlane**
    object
    - **settings.dataPlane.accessLog**
      object
      - **settings.dataPlane.accessLog.jsonLabels**
        object

        Structured access log format using key-value pairs. Template operators are described in envoy documentation.

        Default: `{"start time":"%START TIME%","method":"%REQ(:METHOD)%","path":"%REQ(X-ENVOY-ORIGINAL-PATH?:PATH)%","protocol":"%PROTOCOL%","response_code":"%RESPONSE_CODE%","response_flags":"%RESPONSE FLAGS%","bytes received":"%BYTES_RECEIVED%","bytes_sent":"%BYTES_SENT%","duration":"%DURATION%","user agent":"%REQ(USER-AGENT)%","authority":"%REQ(:AUTHORITY)%","upstream_host":"%UPSTREAM_HOST%"}`

      - **settings.dataPlane.accessLog.textFormat**
        string

        Sidecar's access log format template. Template operators are described in envoy documentation.

        Default: `[%START TIME%] "%REQ(:METHOD)% %REQ(X-ENVOY-ORIGINAL-PATH?:PATH)% %PROTOCOL%" %RESPONSE CODE% %RESPONSE FLAGS% %RESPONSE CODE DETAILS% %CONNECTION TERMINATION DETAILS% "%UPSTREAM TRANSPORT FAILURE REASON%" %BYTES RECEIVED% %BYTES SENT% %DURATION% %RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)% "%REQ(X-FORWARDED-FOR)%" "%REQ(USER-AGENT)%" "%REQ(X-REQUEST-ID)%" "%REQ(:AUTHORITY)%" "%UPSTREAM HOST%" %UPSTREAM LOCAL ADDRESS% %DOWNSTREAM_LOCAL_ADDRESS% %DOWNSTREAM_REMOTE_ADDRESS% %REQUESTED_SERVER_NAME% %ROUTE_NAME%`

      - **settings.dataPlane.accessLog.type**
        string

        Default: `Text`

        Allowed values: `Text`, `JSON`

    - **settings.dataPlane.enableHTTP10**
      boolean

      Whether to handle HTTP/1.0 requests in istio-sidecars or deny them with `426 Upgrade Required` response.

      Default: `false`

      Example:

      ```
      enableHTTP10: true
      ```

    - **settings.dataPlane.extensionProviders**
      array of objects

      Additional Istio extension providers to register in the mesh configuration.

      The provider name `main-access-log-format` is reserved by Deckhouse.

      Default: `[]`

      Example:

      ```
      extensionProviders:
      - name: authservice-grpc
        envoyExtAuthzGrpc:
          service: authservice.example.svc.cluster.local
          port: 10003
      ```

      - **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc**
        object

        Required value

        External authorization provider that implements Envoy ext_authz gRPC API.

        - **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.clearRouteCache**
          boolean

          Clear route cache to let the external authorization service affect routing decisions.

        - **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.failOpen**
          boolean

          Allow requests when the authorization service is unavailable or returns a 5xx error.

        - **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.port**
          integer

          Required value

          Service port.

          Allowed values: `1 <= X <= 65535`

        - **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.service**
          string

          Required value

Service that implements Envoy ext_authz gRPC API. The format is `[<hostname>]/[<namespace>]`.

- **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.statusOnError**
  string

  HTTP status returned to the client when communication with the authorization service fails.

- **settings.dataPlane.extensionProviders.envoyExtAuthzGrpc.timeout**
  string

  Maximum duration the proxy waits for the authorization service response.

- **settings.dataPlane.extensionProviders.name**
  string

  Required value

  Unique extension provider name.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

- **settings.dataPlane.proxyConfig**
  object

  Mesh-wide defaults for ProxyConfig configurations.

  - **settings.dataPlane.proxyConfig.holdApplicationUntilProxyStarts**
    boolean

    With this feature, the sidecar-injector injects the sidecar at the first place of Pod’s container list and adds a postStart hook to be sure if the Envoy proxy is initialized before the application. So the Envoy is able to handle requests without application network errors.

    This global flag can be overriden per Pod by an annotation — `proxy.istio.io/config: '{ "holdApplicationUntilProxyStarts": true }'`.

    Default: `false`

    Example:

    ```
    holdApplicationUntilProxyStarts: true
    ```

  - **settings.dataPlane.proxyConfig.idleTimeout**
    string

    Timeout for connections without application activity established between the client’s istio-sidecar and the service. When the timeout expires, the connection between the sidecar and the service is closed, but the connection between the application and the sidecar is not closed. If set to `0s`, the timeout is disabled.

    This global flag can be overriden per Pod by an annotation:

    ```
    proxy.istio.io/config: |-
      proxyMetadata:
        ISTIO_META_IDLE_TIMEOUT: "12h"
    ```

    > **Warning!** Disabling this timeout (setting the value to `0s`) is very likely to result in leaky connections due to TCP FIN packet loss, etc. **Warning!** After changing this setting, a restart of the client pods is required.

    Default: `1h`

    Pattern: `^[0-9]+(s|m|h)$`

    Example:

    ```
    idleTimeout: 24h
    ```

- **settings.dataPlane.trafficRedirectionSetupMode**
  string

  Managing the redirection mode of application traffic to be forwarded under Istio control in the Pod’s network namespace.

  - `CNIPlugin` — in this mode, the configuration is performed by a CNI plugin when creating a Pod on a node. This mode does not require additional permissions for Pods and is recommended. This mode has limitations when using application init-containers that perform network communication with other services.
  - `InitContainer` — classic mode, each application Pod is automatically injected with a special init-container that configures the network environment of the Pod. In order to perform this configuration, the init-container is given additional permissions, which may not meet the security requirements of individual installations.

  Default: `InitContainer`

  Allowed values: `CNIPlugin`, `InitContainer`

  Examples:

  ```
  trafficRedirectionSetupMode: CNIPlugin
  ```

  ```
  trafficRedirectionSetupMode: InitContainer
  ```

- **settings.dataPlane.ztunnel**
  object

  ztunnel specific settings.

  - **settings.dataPlane.ztunnel.resourcesManagement**
    object

Settings for CPU and memory requests and limits by istiod pods.

Examples:

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: InPlaceOrRecreate
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

```
resourcesManagement:
  mode: Static
  static:
    requests:
      cpu: 55m
      memory: 256Mi
    limits:
      cpu: '2'
      memory: 2Gi
```

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: Auto
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

- **settings.dataPlane.ztunnel.resourcesManagement.mode**
  string

  Resource management mode:

  - `Static` — allows you to specify requests/limits. The parameters of this mode are defined in the static parameter section;
  - `VPA` — uses VPA. You can configure this mode by modifying parameters in the vpa parameter section.

  Default: `VPA`

  Allowed values: `VPA`, `Static`

- **settings.dataPlane.ztunnel.resourcesManagement.static**
  object

  Resource management options for the `Static` mode.

  - **settings.dataPlane.ztunnel.resourcesManagement.static.limits**
    object

    Configuring CPU and memory limits.

    - **settings.dataPlane.ztunnel.resourcesManagement.static.limits.cpu**
      string

      Configuring CPU limits.

      Pattern: `^[0-9]+m?$`

    - **settings.dataPlane.ztunnel.resourcesManagement.static.limits.memory**
      string or number

      Configuring memory limits.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.dataPlane.ztunnel.resourcesManagement.static.requests**
    object

    Resource requests settings for pods.

    - **settings.dataPlane.ztunnel.resourcesManagement.static.requests.cpu**
      string

      Configuring CPU requests.

      Pattern: `^[0-9]+m?$`

    - **settings.dataPlane.ztunnel.resourcesManagement.static.requests.memory**
      string or number

      Configuring memory requests.

Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.dataPlane.ztunnel.resourcesManagement.vpa**
  object

  Resource management options for the `VPA` mode.

  - **settings.dataPlane.ztunnel.resourcesManagement.vpa.cpu**
    object

    CPU-related VPA settings.

    - **settings.dataPlane.ztunnel.resourcesManagement.vpa.cpu.limitRatio**
      number

      The CPU limits/requests ratio.

      This ratio is used for calculating the initial CPU limits for a pod.

      If this parameter is set, the VPA will recalculate the CPU limits while maintaining the specified limits/requests ratio.

    - **settings.dataPlane.ztunnel.resourcesManagement.vpa.cpu.max**
      string or number

      The maximum value that the VPA can set for the CPU requests.

      Default: `2`

      Pattern: `^[0-9]+m?$`

    - **settings.dataPlane.ztunnel.resourcesManagement.vpa.cpu.min**
      string or number

      The minimum value that the VPA can set for the CPU requests.

      Default: `50m`

      Pattern: `^[0-9]+m?$`

  - **settings.dataPlane.ztunnel.resourcesManagement.vpa.memory**
    object

    Memory-related VPA settings.

    - **settings.dataPlane.ztunnel.resourcesManagement.vpa.memory.limitRatio**
      number

      The memory limits/requests ratio.

      This ratio is used for calculating the initial memory limits for a pod.

      If this parameter is set, the VPA will recalculate the memory limits while maintaining the specified limits/requests ratio.

    - **settings.dataPlane.ztunnel.resourcesManagement.vpa.memory.max**
      string or number

      The maximum memory requests the VPA can set.

      Default: `2Gi`

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    - **settings.dataPlane.ztunnel.resourcesManagement.vpa.memory.min**
      string or number

      The minimum memory requests the VPA can set.

      Default: `256Mi`

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.dataPlane.ztunnel.resourcesManagement.vpa.mode**
    string

    VPA operating mode.

    - `Initial` — VPA sets initial values for pod resource requests and limits at pod creation time. Resource values are not changed automatically afterwards.
    - `InPlaceOrRecreate` — VPA attempts to update pod resources in place when supported by the cluster. If in-place updates are not possible, the pod will be recreated.
    - `Auto` — VPA automatically recreates pods to apply updated resource values.

      > **i** Starting from Deckhouse version 1.75, this mode is considered legacy. It is recommended to use `InPlaceOrRecreate` instead.

    Default: `InPlaceOrRecreate`

    Allowed values: `Initial`, `InPlaceOrRecreate`, `Auto`

- **settings.federation**
  object

  Available in editions: EE

  Parameters for federating with other clusters.

  - **settings.federation.enabled**
    boolean

Designate this cluster as a federation member (see Enabling federation).

Default: `false`

Example:

```
enabled: true
```

- **settings.globalVersion**
  string

  Specific version of Istio control-plane which handles unspecific versions of data plane (namespaces with `istio-injection=enabled` label, not `istio.io/rev=`).

  Default: `1.25`

  Pattern: `^[0-9]+\.[0-9]+$`

- **settings.highAvailability**
  boolean

  Manually enable the high availability mode.

  By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

  Example:

  ```
  highAvailability: true
  ```

- **settings.https**
  object

  What certificate type to use with module's public web interfaces.

  This parameter completely overrides the `global.modules.https` settings.

  Examples:

  ```
  https:
    mode: CustomCertificate
    customCertificate:
      secretName: foobar
  ```

  ```
  https:
    mode: CertManager
    certManager:
      clusterIssuerName: letsencrypt
  ```

  - **settings.https.certManager**
    object
    - **settings.https.certManager.clusterIssuerName**
      string

      What ClusterIssuer to use for Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy.

      Currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available. Also, you can define your own.

      Default: `letsencrypt`

  - **settings.https.customCertificate**
    object
    - **settings.https.customCertificate.secretName**
      string

      The name of the secret in the `d8-system` namespace to use with Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy.

      This secret must have the kubernetes.io/tls format.

      Default: `false`

  - **settings.https.mode**
    string

    The HTTPS usage mode:

    - `CertManager` — Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy will use HTTPS and get a certificate from the clusterissuer defined in the `certManager.clusterIssuerName` parameter.
    - `CustomCertificate` — Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy will use HTTPS using the certificate from the `d8-system` namespace.
    - `OnlyInURI` — Kiali/metadata-exporter (including SPIFFE endpoint)/api-proxy will work over HTTP (thinking that there is an external HTTPS load balancer in front that terminates HTTPS traffic). All the links in the `user-authn` will be generated using the HTTPS scheme. Load balancer should provide a redirect from HTTP to HTTPS.

    Caution! Unlike other modules, Istio doesn't support non-secured HTTP (`mode: Disabled`).

    Default: `CertManager`

    Allowed values: `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller used for Kiali, metadata-exporter and proxy-api.

  Optional. By default, the `modules.ingressClass` global value is used.

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.jwksResolverAdditionalRootCA**
  string

  An additional root certificate in PEM format. It is used by istiod when resolving JWKS URIs over HTTPS.

  Example: —–BEGIN CERTIFICATE—– ... —–END CERTIFICATE—–

- **settings.multicluster**
  object

  Available in editions: EE

  Multicluster parameters.

  - **settings.multicluster.enabled**
    boolean

    Designate this cluster as a multicluster member (see Enabling multicluster).

    Default: `false`

    Example:

    ```
    enabled: true
    ```

- **settings.nodeSelector**
  object

  Optional `nodeSelector` for istio-operator, metadata-exporter and Kiali. The same as the `spec.nodeSelector` pod parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

- **settings.outboundTrafficPolicyMode**
  string

  How to handle requests directed to external services which aren’t registered in service mesh.

  - `AllowAny` — allow.
  - `RegistryOnly` — deny. In this case to work with external services you need to register them with ServiceEntry custom resource or to organize egressgateway.

  Default: `AllowAny`

  Allowed values: `AllowAny`, `RegistryOnly`

  Example:

  ```
  outboundTrafficPolicyMode: AllowAny
  ```

- **settings.sidecar**
  object

  Network settings for traffic capture by istio sidecar.

  - **settings.sidecar.excludeInboundPorts**
    array of strings

    The range of inbound ports whose traffic is guaranteed not to flow through Istio.

    You can redefine this parameter for single Pod using the `traffic.sidecar.istio.io/excludeInboundPorts` annotation.

    Default: `[]`

    Example:

    ```
    excludeInboundPorts:
    - '8080'
    - '8443'
    ```

    - **Element of the array**
      string

      Pattern: `^[0-9]{1,5}$`

  - **settings.sidecar.excludeOutboundIPRanges**
    array of strings

    Traffic to these IP ranges is guaranteed not to flow through Istio.

    You can redefine this parameter for single Pod using the `traffic.sidecar.istio.io/excludeOutboundIPRanges` annotation.

    Default: `[]`

    Example:

    ```
    excludeOutboundIPRanges:
    - 10.1.1.0/24
    ```

    - **Element of the array**
      string

      Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

  - **settings.sidecar.excludeOutboundPorts**
    array of strings

The range of outbound ports whose traffic is guaranteed not to flow through Istio.

You can redefine this parameter for single Pod using the `traffic.sidecar.istio.io/excludeOutboundPorts` annotation.

Default: `[]`

Example:

```
excludeOutboundPorts:
- '8080'
- '8443'
```

- **Element of the array**
  string

  Pattern: `^[0-9]{1,5}$`

- **settings.sidecar.includeOutboundIPRanges**
  array of strings

  Traffic to these IP ranges is forcibly routed through Istio.

  You can redefine this parameter for single Pod using the `traffic.sidecar.istio.io/includeOutboundIPRanges` annotation.

  Default: `["0.0.0.0/0"]`

  Example:

  ```
  includeOutboundIPRanges:
  - 10.1.1.0/24
  ```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **settings.sidecar.resourcesManagement**
  object

  Manages Istio sidecar container resources.

  Caution! The setting only applies to new Pods with `istio-proxy`.

  Example:

  ```
  resourcesManagement:
    static:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 2000m
        memory: 1Gi
  ```

  - **settings.sidecar.resourcesManagement.mode**
    string

    Resource management mode:

    - `Static` — allows you to specify requests/limits. The parameters of this mode are defined in the static parameter section;

    Default: `Static`

    Allowed values: `Static`

  - **settings.sidecar.resourcesManagement.static**
    object

    Resource management options for the `Static` mode.

    - **settings.sidecar.resourcesManagement.static.limits**
      object

      Configuring CPU and memory limits.

      - **settings.sidecar.resourcesManagement.static.limits.cpu**
        string

        Configuring CPU limits.

        Default: `2000m`

        Pattern: `^[0-9]+m?$`

      - **settings.sidecar.resourcesManagement.static.limits.memory**
        string or number

        Configuring memory limits.

        Default: `1Gi`

        Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    - **settings.sidecar.resourcesManagement.static.requests**
      object

Resource requests settings for pods.

- **settings.sidecar.resourcesManagement.static.requests.cpu**
  string

  Configuring CPU requests.

  Default: `100m`

  Pattern: `^[0-9]+m?$`

- **settings.sidecar.resourcesManagement.static.requests.memory**
  string or number

  Configuring memory requests.

  Default: `128Mi`

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.tolerations**
  array of objects

  Optional `tolerations` for istio-operator, metadata-exporter and Kiali. The same as `spec.tolerations` for the Kubernetes pod.

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string
- **settings.tracing**
  object

  Tracing parameters.

  - **settings.tracing.collector**
    object

    Tracing collection settings.

    - **settings.tracing.collector.zipkin**
      object

      Zipkin protocol parameters used by Istio for sending traces. Jaeger supports this protocol.

      If tracing is enabled, this settings section is mandatory.

      - **settings.tracing.collector.zipkin.address**
        string

        Network address of zipkin collector in `<IP of FQDN>:<port>` format.

        Pattern: `[0-9a-zA-Z\.-]+`

        Example:

        ```
        address: zipkin.myjaeger.svc:9411
        ```
  - **settings.tracing.enabled**
    boolean

    Turn on or off tracing collection and displaying in Kiali.

    Default: `false`

    Example:

    ```
    enabled: true
    ```
  - **settings.tracing.kiali**
    object

    Span displaying settings for Kiali.

    When not configured, Kiali won’t show any tracing dashboards.

    Examples:

    ```
    kiali: {}
    ```

    ```
    kiali:
      jaegerURLForUsers: https://tracing-service:4443/jaeger
      jaegerGRPCEndpoint: http://tracing.myjaeger.svc:16685/
    ```

    - **settings.tracing.kiali.jaegerGRPCEndpoint**
      string

      Accessible from cluster address of jaeger GRPC interface for system queries by Kiali.

      When not configured, Kiali will only show external links using the `jaegerURLForUsers` config without interpretationing.

Example:

```
jaegerGRPCEndpoint: http://tracing.myjaeger.svc:16685/
```

- **settings.tracing.kiali.jaegerURLForUsers**
  string

  Jaeger UI address for users. Mandatory parameter if Kiali is enabled.

  Example:

  ```
  jaegerURLForUsers: https://tracing-service:4443/jaeger
  ```

- **settings.tracing.sampling**
  number

  The sampling rate option can be used to control what percentage of requests get reported to your tracing system.

  This should be configured depending upon your traffic in the mesh and the amount of tracing data you want to collect.

  It is possible to override this option with the following Pod annotation:

  ```
  proxy.istio.io/config: |
    tracing:
      sampling: 100.0
  ```

  Default: `1.0`

  Allowed values: `0.01 <= X <= 100.0`

  Example:

  ```
  sampling: 50.05
  ```

Authentication

user-authn module provides authentication by default. Also, externalAuthentication can be configured (see below). If these options are disabled, the module will use basic auth with the auto-generated password.

Use d8 k to see password:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values istio -o json | jq
'.istio.internal.auth.password'
```

Delete the Secret to re-generate password:

```
d8 k -n d8-istio delete secret/kiali-basic-auth
```

⚠ The `auth.password` parameter is deprecated.

Custom Resources

i The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

i The module lifecycle stage: General Availability

IngressIstioController

Scope: Cluster
Version: v1alpha1

- **spec**
  object

  Required value

  - **spec.hostPort**
    object

    `HostPort` inlet settings.

    - **spec.hostPort.httpPort**
      integer

      Port for insecure HTTP connections.

      If the parameter is not set, the connection over HTTP cannot be established.

      This parameter is mandatory if `httpsPort` is not set.

      Example:

      ```
      httpPort: 80
      ```

- **spec.hostPort.httpsPort**
  integer

  Port for secure HTTPS connections.

  If the parameter is not set, the connection over HTTPS cannot be established.

  This parameter is mandatory if `httpPort` is not set.

  Example:

  ```
  httpsPort: 443
  ```

- **spec.ingressGatewayClass**
  string

  Required value

  Ingress gateway class is used by application Gateway resources for identifying the right Ingress gateway setup.

  The identification is organized by setting the spec.selector: `istio.deckhouse.io/ingress-gateway-class: <ingressGatewayClass value>`.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

  Example:

  ```
  ingressGatewayClass: istio
  ```

- **spec.inlet**
  string

  Required value

  The way traffic goes to cluster from the outer network.

  - `LoadBalancer` — Ingress controller is deployed and the service of `LoadBalancer` type is provisioned.
  - `NodePort` — Ingress controller is deployed and available through nodes' ports via `nodePort`.
  - `HostPort` — Ingress controller is deployed and available through nodes' ports via `hostPort`.

    Caution! There can be only one controller with this inlet type on a host.

  Allowed values: `LoadBalancer`, `HostPort`, `NodePort`

- **spec.loadBalancer**
  object

  Not required value.

  A section of parameters of the `LoadBalancer` inlet.

  - **spec.loadBalancer.annotations**
    object

    Annotations to assign to the service for flexible configuration of the load balancer.

- **spec.nodePort**
  object

  `HostPort` inlet settings.

  - **spec.nodePort.httpPort**
    integer

    Port for insecure HTTP connections.

    If the parameter is not set, the connection over HTTP cannot be established.

    This parameter is mandatory if `httpsPort` is not set.

    Example:

    ```
    httpPort: 30080
    ```

  - **spec.nodePort.httpsPort**
    integer

    Port for secure HTTPS connections.

    If the parameter is not set, the connection over HTTPS cannot be established.

    This parameter is mandatory if `httpPort` is not set.

    Example:

    ```
    httpsPort: 30443
    ```

- **spec.nodeSelector**
  object

  The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Format: the standard `nodeSelector` list. Instance pods inherit this field as is.

- **spec.resourcesRequests**
  object

  Max amounts of CPU and memory resources that the pod can request when selecting a node (if the VPA is disabled, then these

values become the default ones).

- **spec.resourcesRequests.mode**
  string

  Required value

  The mode for managing resource requests.

  Default: `VPA`

  Allowed values: `VPA`, `Static`
- **spec.resourcesRequests.static**
  object

  Static mode settings.

  - **spec.resourcesRequests.static.cpu**
    string

    CPU requests.

    Default: `350m`
  - **spec.resourcesRequests.static.memory**
    string

    Memory requests.

    Default: `500Mi`
- **spec.resourcesRequests.vpa**
  object

  Parameters of the vpa mode.

  - **spec.resourcesRequests.vpa.cpu**
    object

    CPU-related parameters.

    - **spec.resourcesRequests.vpa.cpu.max**
      string

      Maximum allowed CPU requests.

      Default: `100m`
    - **spec.resourcesRequests.vpa.cpu.min**
      string

      Minimum allowed CPU requests.

      Default: `50m`
  - **spec.resourcesRequests.vpa.memory**
    object

    The amount of memory requested.

    - **spec.resourcesRequests.vpa.memory.max**
      string

      Maximum allowed memory requests.

      Default: `200Mi`
    - **spec.resourcesRequests.vpa.memory.min**
      string

      Minimum allowed memory requests.

      Default: `100Mi`
  - **spec.resourcesRequests.vpa.mode**
    string

    The VPA usage mode.

    Default: `Initial`

    Allowed values: `Initial`, `Auto`

- **spec.tolerations**
  array of objects

  The same as in the pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  Format: the standard toleration list. Instance pods inherit this field as is.

  - **spec.tolerations.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`
  - **spec.tolerations.key**
    string
  - **spec.tolerations.operator**
    string

    Default: `Equal`

Allowed values: `Exists`, `Equal`

- **spec.tolerations.tolerationSeconds**
  integer
- **spec.tolerations.value**
  string

IstioFederation

Scope: Cluster
Version: v1alpha1

Custom resource for setting remote cluster as trusted one.

- **spec**
  object

  Required value

  Available in editions: EE

  - **spec.metadata**
    object
    - **spec.metadata.ca**
      string

      Available in editions: EE

      Certificate for validation HTTPS endpoint with remote cluster metadata.

      Example:

      ```
      ca: "-----BEGIN CERTIFICATE----- ..... -----END CERTIFICATE-----"
      ```
    - **spec.metadata.insecureSkipVerify**
      boolean

      Available in editions: EE

      Skip validation certificate on HTTPS endpoint with remote cluster metadata.

      Default: `false`

      Example:

      ```
      insecureSkipVerify: true
      ```
  - **spec.metadataEndpoint**
    string

    Required value

    Available in editions: EE

    HTTPS endpoint with remote cluster metadata.

    Example:

    ```
    metadataEndpoint: https://istio.k8s.example.com/metadata/
    ```
  - **spec.trustDomain**
    string

    Required value

    Available in editions: EE

    The TrustDomain of the remote cluster.

    A mandatory parameter, but it is not currently utilized, as Istio does not yet support mapping TrustDomain to a root Certificate Authority (CA).

    Pattern: `^[0-9a-zA-Z._-]+$`

    Example:

    ```
    trustDomain: cluster.local
    ```
- **status**
  object
  - **status.metadataCache**
    object
    - **status.metadataCache.private**
      object
      - **status.metadataCache.private.ingressGateways**
        array of objects
        - **status.metadataCache.private.ingressGateways.address**
          string
        - **status.metadataCache.private.ingressGateways.port**
          integer
      - **status.metadataCache.private.publicServices**
        array of objects
        - **status.metadataCache.private.publicServices.hostname**
          string
        - **status.metadataCache.private.publicServices.ports**
          array of objects
          - **status.metadataCache.private.publicServices.ports.name**
            string

                - **status.metadataCache.private.publicServices.ports.port**
                  integer
                - **status.metadataCache.private.publicServices.ports.protocol**
                  string
        - **status.metadataCache.privateLastFetchTimestamp**
          string
        - **status.metadataCache.public**
          object
            - **status.metadataCache.public.authnKeyPub**
              string
            - **status.metadataCache.public.clusterUUID**
              string
            - **status.metadataCache.public.rootCA**
              string
        - **status.metadataCache.publicLastFetchTimestamp**
          string

IstioMulticluster

Scope: Cluster
Version: v1alpha1

Custom resource for setting remote cluster as trusted one.

- **spec**
  object

  Required value

  Available in editions: EE

  - **spec.enableIngressGateway**
    boolean

    Available in editions: EE

    Using IngressGateway for accessing remote Pods.

    If remote Pods are accessible directly from our cluster (flat network), it is efficient not to use extra hop.

    Default: `true`

  - **spec.metadata**
    object
    - **spec.metadata.ca**
      string

      Available in editions: EE

      HTTPS certificate authority for remote cluster metadata.

    - **spec.metadata.insecureSkipVerify**
      boolean

      Available in editions: EE

      HTTPS check or not for remote cluster metadata.

      Default: `false`

      Example:

      ```
      insecureSkipVerify: true
      ```

  - **spec.metadataEndpoint**
    string

    Required value

    Available in editions: EE

    HTTPS endpoint with remote cluster metadata.

    Example:

    ```
    metadataEndpoint: https://istio.k8s.example.com/metadata/
    ```

- **status**
  object
  - **status.metadataCache**
    object
    - **status.metadataCache.private**
      object
      - **status.metadataCache.private.apiHost**
        string
      - **status.metadataCache.private.ca**
        string
      - **status.metadataCache.private.ingressGateways**
        array of objects
        - **status.metadataCache.private.ingressGateways.address**
          string
        - **status.metadataCache.private.ingressGateways.port**
          integer
      - **status.metadataCache.private.networkName**
        string
    - **status.metadataCache.privateLastFetchTimestamp**
      string
    - **status.metadataCache.public**

object
- **status.metadataCache.public.authnKeyPub**
  string
- **status.metadataCache.public.clusterUUID**
  string
- **status.metadataCache.public.rootCA**
  string

- **status.metadataCache.publicLastFetchTimestamp**
  string

Custom Resources (by istio.io)

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

Routing

DestinationRule

Reference.

Allows you to:

- Define a strategy for balancing traffic between service endpoints:
  - The balancing algorithm to use (LEAST_CONN, ROUND_ROBIN, ...);
  - Symptoms that an endpoint is unhealthy and rules for removing it from load balancing;
  - Limits for TCP connections and endpoint requests;
  - Sticky Sessions;
  - Circuit Breaker;
- Define alternative endpoint groups for traffic processing (suitable for Canary Deployments). Also, you can configure balancing strategies for each group;
- Configuring TLS for outgoing requests.

VirtualService

Reference.

Using VirtualService is optional; regular services fit just fine if their capabilities match your requirements.

Allows you to configure request routing:

- Arguments on which routing decisions are based on:
  - Host;
  - URI;
  - Weight;
- Parameters of the resulting directions:
  - The new host;
  - The new URI;
  - If the host is defined using DestinationRule, then requests can be sent to `subsets`;
  - Timeout and retry settings.

> **⚠** Istio must be aware of the `destination`; if you use an external API, register it via ServiceEntry.

ServiceEntry

Reference.

It is similar to Endpoints + Service in vanilla Kubernetes. Informs Istio about the existence of an external service and lets you redefine its address.

Authentication

Answers the question “Who has made the request?” Not to be confused with authorization - the latter determines what the authenticated subject can or cannot do.

There are two authentication methods:

- mTLS;
- JWT tokens;

PeerAuthentication

Reference.

Allows you to define the mTLS strategy for an individual namespace. Defines how traffic will be tunneled (or not) to the sidecar. Each mTLS request can automatically identify the source and allows you to use it in the authorization rules.

RequestAuthentication

Reference.

Allows you to configure JWT authentication for requests.

Authorization

⚠ Authorization without the use of mTLS or JWT authentication will not work fully. In this case, you will be able to use only basic arguments, such as `source.ip` and `request.headers`, for defining policies.

AuthorizationPolicy

Reference.

Enables and defines access control to the `workload`. The `AuthorizationPolicy` custom resource supports both ALLOW and DENY rules. The following decision-making algorithm is used if at least one policy is defined for a workload:

- The request is denied if it falls under the DENY policy.
- The request is allowed if there are no ALLOW policies for the service.
- The request is allowed if it falls under the ALLOW policy.
- In all other cases, the request is denied.

The following arguments are used in the decision-making algorithm:

- source:
  - namespace
  - principal (i.e., the ID the user has received after authentication)
  - IP
- destination:
  - method (GET, POST, ...)
  - Host
  - port
  - URI
- conditions:
  - HTTP-headers
  - source arguments
  - destination arguments
  - JWT-tokens

Sidecar

Reference

This resource limits the number of services for which information is transmitted to the `istio-proxy` sidecar.

Examples

ℹ The module lifecycle stage: General Availability

Circuit Breaker

The `outlierDetection` settings in the DestinationRule custom resource help to determine whether some endpoints do not behave as expected. Refer to the Envoy documentation for more details on the Outlier Detection algorithm.

Example:

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: reviews-cb-policy
spec:
  host: reviews.prod.svc.cluster.local
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 100 # The maximum number of connections to the host (cumulative for all endpoints)
      http:
        maxRequestsPerConnection: 10 # The connection will be re-established after every 10 requests
    outlierDetection:
      consecutive5xxErrors: 7 # Seven consecutive errors are allowed (including 5XX, TCP and HTTP timeouts)
      interval: 5m          # over 5 minutes.
      baseEjectionTime: 15m  # Upon reaching the error limit, the endpoint will be excluded from balancing for 15
minutes.
```

Additionally, the VirtualService resource is used to configure the HTTP timeouts. These timeouts are also taken into account when calculating error statistics for endpoints.

Example:

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: my-productpage-rule
  namespace: myns
spec:
  hosts:
  - productpage
  http:
  - timeout: 5s
    route:
    - destination:
       host: productpage
```

gRPC balancing

⚠ Assign a name with the `grpc` prefix or value to the port in the corresponding service to make gRPC service balancing start automatically.

Locality Failover

i Read the main documentation if you need.

Istio allows you to configure a priority-based locality (geographic location) failover between endpoints. Istio uses node labels with the appropriate hierarchy to define the zone:

- `topology.istio.io/subzone`
- `topology.kubernetes.io/zone`
- `topology.kubernetes.io/region`

This comes in handy for inter-cluster failover when used together with a multicluster.

⚠ The Locality Failover can be enabled using the DestinationRule CR. Note that you also have to configure the outlierDetection.

Example:

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: helloworld
spec:
  host: helloworld
  trafficPolicy:
    loadBalancer:
      localityLbSetting:
        enabled: true # Locality Failover is enabled
    outlierDetection: # outlierDetection must be enabled
      consecutive5xxErrors: 1
      interval: 1s
      baseEjectionTime: 1m
```

Retry

You can use the VirtualService resource to configure Retry for requests.

⚠ All requests (including POST ones) are retried three times by default.

Example:

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: ratings-route
spec:
  hosts:
  - ratings.prod.svc.cluster.local
  http:
  - route:
    - destination:
       host: ratings.prod.svc.cluster.local
    retries:
      attempts: 3
      perTryTimeout: 2s
      retryOn: gateway-error,connect-failure,refused-stream
```

Canary

⚠ Istio is only responsible for flexible request routing that relies on special request headers (such as cookies) or simply randomness. The CI/CD system is responsible for customizing this routing and “switching” between canary versions.

The idea is that two Deployments with different versions of the application are deployed in the same namespace. The Pods of different versions have different labels (`version: v1` and `version: v2`).

You have to configure two custom resources:

- A DestinationRule – defines how to identify different versions of your application (subsets);
- A VirtualService – defines how to balance traffic between different versions of your application.

Example:

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: productpage-canary
spec:
  host: productpage
  # subsets are only available when accessing the host via the VirtualService from a Pod managed by Istio.
  # These subsets must be defined in the routes.
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

Cookie-based routing

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: productpage-canary
spec:
  hosts:
  - productpage
  http:
  - match:
    - headers:
       cookie:
         regex: "^(.*;?)?(canary=yes)(;.*)?"
    route:
    - destination:
        host: productpage
        subset: v2 # The reference to the subset from the DestinationRule.
  - route:
    - destination:
        host: productpage
        subset: v1
```

Probability-based routing

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: productpage-canary
spec:
  hosts:
  - productpage
  http:
  - route:
    - destination:
        host: productpage
        subset: v1 # The reference to the subset from the DestinationRule.
      weight: 90 # Percentage of traffic that the Pods with the version: v1 label will be getting.
  - route:
    - destination:
        host: productpage
        subset: v2
      weight: 10
```

Ingress to publish applications

Istio Ingress Gateway

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: IngressIstioController
metadata:
 name: main
spec:
  # ingressGatewayClass contains the label selector value used to create the Gateway resource
  ingressGatewayClass: istio-hp
  inlet: HostPort
  hostPort:
    httpPort: 80
    httpsPort: 443
  nodeSelector:
    node-role.deckhouse.io/frontend: ""
  tolerations:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      operator: Equal
      value: frontend
  resourcesRequests:
    mode: VPA
```

```
apiVersion: v1
kind: Secret
metadata:
  name: app-tls-secret
  namespace: d8-ingress-istio # note the namespace isn't app-ns
type: kubernetes.io/tls
data:
  tls.crt: |
    <tls.crt data>
  tls.key: |
    <tls.key data>
```

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: gateway-app
  namespace: app-ns
spec:
  selector:
    # label selector for using the Istio Ingress Gateway main-hp
    istio.deckhouse.io/ingress-gateway-class: istio-hp
  servers:
    - port:
        # standard template for using the HTTP protocol
        number: 80
        name: http
        protocol: HTTP
      hosts:
        - app.example.com
    - port:
        # standard template for using the HTTPS protocol
        number: 443
        name: https
        protocol: HTTPS
      tls:
        mode: SIMPLE
        # a secret with a certificate and a key, which must be created in the d8-ingress-istio namespace
        # supported secret formats can be found at https://istio.io/latest/docs/tasks/traffic-
management/ingress/secure-ingress/#key-formats
        credentialName: app-tls-secret
      hosts:
        - app.example.com
```

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: vs-app
  namespace: app-ns
spec:
  gateways:
    - gateway-app
  hosts:
    - app.example.com
  http:
    - route:
        - destination:
            host: app-svc
```

Ingress NGINX

To use Ingress, you need to:

- Configure the Ingress controller by adding Istio sidecar to it. In our case, you need to enable the `enableIstioSidecar` parameter in the ingress-nginx module's IngressNginxController custom resource.
- Set up an Ingress that refers to the Service. The following annotations are mandatory for Ingress:
  - `nginx.ingress.kubernetes.io/service-upstream: "true"` — using this annotation, the Ingress controller sends requests to a single ClusterIP (from Service CIDR) while envoy load balances them. Ingress controller's sidecar is only catching traffic directed to Service CIDR.
  - `nginx.ingress.kubernetes.io/upstream-vhost: myservice.myns.svc` — using this annotation, the sidecar container can identify the application service that serves requests.

Examples:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: productpage
  namespace: bookinfo
  annotations:
    # Nginx proxies traffic to the ClusterIP instead of pods' own IPs.
    nginx.ingress.kubernetes.io/service-upstream: "true"
    # In Istio, all routing is carried out based on the `Host:` headers.
    # Instead of letting Istio know about the `productpage.example.com` external domain,
    # we use the internal domain of which Istio is aware.
    nginx.ingress.kubernetes.io/upstream-vhost: productpage.bookinfo.svc
spec:
  rules:
    - host: productpage.example.com
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: productpage
              port:
                number: 9080
```

```
apiVersion: v1
kind: Service
metadata:
  name: productpage
  namespace: bookinfo
spec:
  ports:
  - name: http
    port: 9080
  selector:
    app: productpage
  type: ClusterIP
```

Authorization configuration examples

Decision-making algorithm

The following algorithm for deciding the fate of a request becomes active after `AuthorizationPolicy` is created for the application:

- The request is denied if it falls under the DENY policy;
- The request is allowed if there are no ALLOW policies for the application;
- The request is allowed if it falls under the ALLOW policy.
- All other requests are denied.

In other words, if you explicitly deny something, then only this restrictive rule will work. If you explicitly allow something, only explicitly authorized requests will be allowed (however, restrictions will stay in force and have precedence).

⚠ The policies based on high-level parameters like namespace or principal require enabling Istio for all involved applications. Also, there must be organized Mutual TLS between applications.

Examples:

- Let's deny POST requests for the myapp application. Since a policy is defined, only POST requests to the application are denied (as per the algorithm above).

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-post-requests
  namespace: foo
spec:
  selector:
    matchLabels:
      app: myapp
  action: DENY
  rules:
  - to:
    - operation:
        methods: ["POST"]
```

- Below, the ALLOW policy is defined for the application. It only allows requests from the `bar` namespace (other requests are denied).

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: foo
spec:
  selector:
    matchLabels:
      app: myapp
  action: ALLOW # The default value, can be skipped.
  rules:
  - from:
    - source:
        namespaces: ["bar"]
```

- Below, the ALLOW policy is defined for the application. Note that it does not have any rules, so not a single request matches it (still, the policy exists). Thus, our decision-making algorithm suggests that if something is allowed, then everything else is denied. In this case, “everything else” includes all the requests.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: foo
spec:
  selector:
    matchLabels:
      app: myapp
  action: ALLOW # The default value, can be skipped.
  rules: []
```

- Below, the (default) ALLOW policy is defined for the application. Note that it has an empty rule. Any request matches this rule, so it is naturally approved.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all
  namespace: foo
spec:
  selector:
    matchLabels:
      app: myapp
  rules:
  - {}
```

Deny all actions for the foo namespace

There are two ways you can do that:

- Explicitly. Here, the DENY policy is created. It has a single `{}` rule that covers all the requests:

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: foo
spec:
  action: DENY
  rules:
  - {}
```

- Implicitly. Here, the (default) ALLOW policy is created that does not have any rules. Thus, no requests will match it, and the policy will deny all of them.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: foo
spec: {}
```

Deny requests from the foo NS only

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: deny-from-ns-foo
 namespace: myns
spec:
 action: DENY
 rules:
 - from:
   - source:
      namespaces: ["foo"]
```

Allow requests for the foo NS only

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allow-intra-namespace-only
 namespace: foo
spec:
 action: ALLOW
 rules:
 - from:
   - source:
      namespaces: ["foo"]
```

Allow requests from anywhere in the cluster

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allow-all-from-my-cluster
 namespace: myns
spec:
 action: ALLOW
 rules:
 - from:
   - source:
      principals: ["mycluster.local/*"]
```

Allow any requests for foo or bar clusters

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allow-all-from-foo-or-bar-clusters-to-ns-baz
 namespace: baz
spec:
 action: ALLOW
 rules:
 - from:
   - source:
      principals: ["foo.local/*", "bar.local/*"]
```

Allow any requests only from entities in the baz namespace of the foo or bar clusters

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allow-all-from-foo-or-bar-clusters-to-ns-baz
 namespace: baz
spec:
 action: ALLOW
 rules:
 - from:
   - source: # Logical conjunction is used for the rules below.
       namespaces: ["baz"]
       principals: ["foo.local/*", "bar.local/*"]
```

Allow from any cluster (via mTLS)

⚠ The denying rules (if they exist) have priority over any other rules. See the algorithm.

Example:

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allow-all-from-any-cluster-with-mtls
 namespace: myns
spec:
 action: ALLOW
 rules:
 - from:
   - source:
       principals: ["*"] # To force the mTLS usage.
```

Allow all requests from anywhere (including no mTLS - plain text traffic)

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: allow-all-from-any
 namespace: myns
spec:
 action: ALLOW
 rules: [{}]
```

Setting up federation for two clusters using the IstioFederation CR

Cluster A:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioFederation
metadata:
  name: cluster-b
spec:
  metadataEndpoint: https://istio.k8s-b.example.com/metadata/
  trustDomain: cluster-b.local
```

Cluster B:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioFederation
metadata:
  name: cluster-a
spec:
  metadataEndpoint: https://istio.k8s-a.example.com/metadata/
  trustDomain: cluster-a.local
```

Setting up multicluster for two clusters using the IstioMulticluster CR

Cluster A:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioMulticluster
metadata:
  name: cluster-b
spec:
  metadataEndpoint: https://istio.k8s-b.example.com/metadata/
```

Cluster B:

```
apiVersion: deckhouse.io/v1alpha1
kind: IstioMulticluster
metadata:
  name: cluster-a
spec:
  metadataEndpoint: https://istio.k8s-a.example.com/metadata/
```

Control the data-plane behavior

Prevent istio-proxy from terminating before the main application's connections are closed

By default, during termination, all containers in a Pod, including istio-proxy one, receive SIGTERM signal simultaneously. But some applications need time to properly handle the termination and sometimes they need to do some network requests. It isn't possible when the istio-proxy stops before the application do. The solution is to add a preStop hook which evaluates the application's activity via discovering application's network sockets and let the sidecar stop when they aren't in the network namespace.

The annotation below adds the preStop hook to istio-proxy container in application's Pod:

```
annotations:
  inject.istio.io/templates: "sidecar,d8-hold-istio-proxy-termination-until-application-stops"
```

`CNIPlugin` application traffic redirection mode restrictions

Unlike the `InitContainer` mode, the redirection setting is done at the moment of Pod creating, not at the moment of triggering the `istio-init` init-container. This means that application init-containers will not be able to interact with other services because all traffic will be redirected to the `istio-proxy` sidecar container, which is not yet running. Workarounds:

- Run the application init container from the user with uid `1337`. Requests from this user are not intercepted under Istio control.
- Exclude a service IP address or port from Istio control using the `traffic.sidecar.istio.io/excludeOutboundIPRanges` or `traffic.sidecar.istio.io/excludeOutboundPorts` annotations.

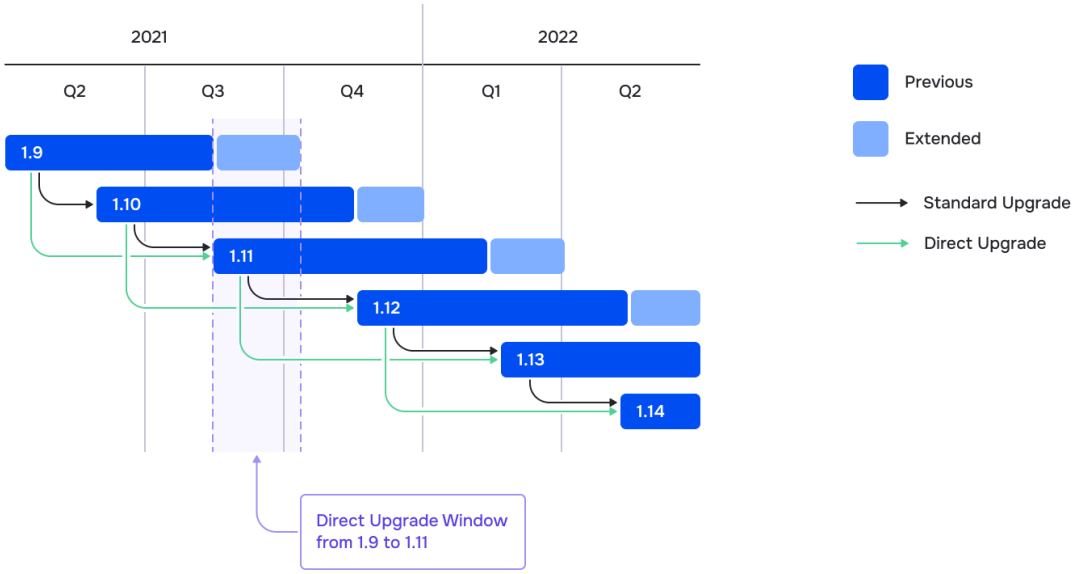
⚠ Each of the workarounds removes traffic from Istio's control, which in turn removes encryption of traffic between application services.

⚠ UID `1337` is reserved by Istio for the `istio-proxy` sidecar container. Do not run your application containers with this UID — their traffic will bypass Istio completely (no routing rules, mTLS, or telemetry will be applied). Use UID `1337` only in init containers when you need to perform network requests before the sidecar is ready.

Upgrading Istio

Upgrading Istio control-plane

- Deckhouse allows you to install different control-plane versions simultaneously:
  - A single global version to handle namespaces or Pods with indifferent version (namespace label `istio-injection: enabled`). It is configured by the globalVersion parameter.
  - Additional versions handle namespaces or Pods with explicitly configured versions (`istio.io/rev: v1x25` label for namespace or Pod). They are configured by the `additionalVersions` parameter.
- Istio declares backward compatibility between data-plane and control-plane in the range of two minor versions:



- Upgrade algorithm (i.e. from `1.21` to `1.25`):
  - Configure additional version in the additionalVersions parameter (`additionalVersions: ["1.25"]`).
- Wait for the corresponding pod `istiod-v1x25-xxx-yyy` to appear in `d8-istio` namespace.
- For every application namespase with istio enabled:
  - Change `istio-injection: enabled` label to `istio.io/rev: v1x25`.
  - Recreate the Pods in namespace (one at a time), simultaneously monitoring the application's workability.
- Reconfigure `globalVersion` to `1.25` and remove the `additionalVersions` configuration.
- Make sure, the old `istiod` pod has gone.

- Change application namespace labels to `istio-injection: enabled`.

To find all Pods with old Istio revision (in the example — version 21), execute the command:

```
d8 k get pods -A -o json | jq --arg revision "v1x21" \
  '.items[] | select(.metadata.annotations."sidecar.istio.io/status" // "{}" | fromjson |
   .revision == $revision) | .metadata.namespace + "/" + .metadata.name'
```

⚠ Upgrading to Istio 1.25 is only possible from version 1.21.

Auto upgrading istio data-plane

To automate istio-sidecar upgrading, set a label `istio.deckhouse.io/auto-upgrade="true"` on the application `Namespace` or on the individual resources — `Deployment`, `DaemonSet` or `StatefulSet`.

Customizing istio-proxy sidecar resource management

You can override the global istio-proxy sidecar resource limits for specific workloads by adding annotations to your application Pods.

Supported annotations

Use these Pod annotations to customize sidecar resources:

Annotation	Description	Example Value
`sidecar.istio.io/proxyCPU`	CPU request for sidecar	`200m`
`sidecar.istio.io/proxyCPULimit`	CPU limit for sidecar	`"1"`
`sidecar.istio.io/proxyMemory`	Memory request for sidecar	`128Mi`
`sidecar.istio.io/proxyMemoryLimit`	Memory limit for sidecar	`512Mi`

Configuration Examples

For Deployments:

```
apiVersion: apps/v1
kind: Deployment
metadata:
# ...
spec:
  template:
    metadata:
      annotations:
         sidecar.istio.io/proxyCPU: 200m
         sidecar.istio.io/proxyCPULimit: "1"
         sidecar.istio.io/proxyMemory: 128Mi
         sidecar.istio.io/proxyMemoryLimit: 512Mi
# ... rest of your deployment spec
```

For ReplicaSets:

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
# ...
spec:
  template:
    metadata:
      annotations:
         sidecar.istio.io/proxyCPU: 200m
         sidecar.istio.io/proxyCPULimit: "1"
         sidecar.istio.io/proxyMemory: 128Mi
         sidecar.istio.io/proxyMemoryLimit: 512Mi
# ... rest of your deployment spec
```

For Pod:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    sidecar.istio.io/proxyCPU: 200m
    sidecar.istio.io/proxyCPULimit: "1"
    sidecar.istio.io/proxyMemory: 128Mi
    sidecar.istio.io/proxyMemoryLimit: 512Mi
# ... rest of your pod spec
```

⚠ All four parameters must be defined together - if you set any of these annotations, you must specify all four (`sidecar.istio.io/proxyCPU`, `sidecar.istio.io/proxyCPULimit`, `sidecar.istio.io/proxyMemory`, and `sidecar.istio.io/proxyMemoryLimit`).

The keepalived module

i The module lifecycle stage: General Availability

⚠ This module does not work with the cilium module.

This module (managed via custom resources) is intended for configuring keepalived clusters on nodes.

Third-party components

List of third-party software used in the `keepalived` module:

- keepalived 2.2.7

  License: GNU General Public License v2.0

  Provides simple and robust facilities for loadbalancing and high-availability to Linux system and Linux based infrastructures.

Configuration

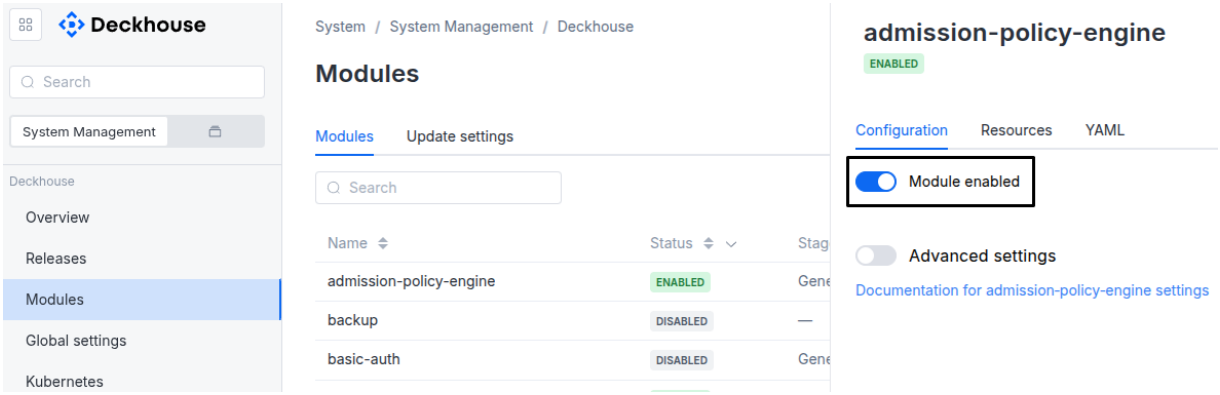
i The module lifecycle stage: General Availability

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `keepalived` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable keepalived
```

- **Using ModuleConfig `keepalived`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `keepalived` (create it if necessary);

  Example of a manifest to enable module `keepalived`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: keepalived
spec:
  enabled: true
```

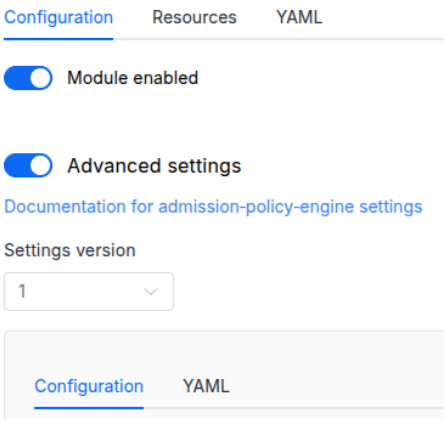
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `keepalived` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `keepalived` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `keepalived`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `keepalived` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc keepalived
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `keepalived` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `keepalived`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: keepalived
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Keepalived clusters are configured using the custom resource.

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

KeepalivedInstance

Scope: Cluster
Version: v1alpha1

The custom resource for configuring keepalived clusters.

Each `KeepalivedInstance` custom resource defines a single keepalived cluster.

`metadata.name` — the name of the cluster (used in pod names).

- **spec**
  object

  Required value

  - **spec.nodeSelector**
    object

    Required value

    Defines the `nodeSelector` parameter for keepalived pods.

    Format: the standard `nodeSelector` list. Instance pods inherit this field as is.

  - **spec.tolerations**
    array of objects

Defines `tolerations` for keepalived pods.

Format: the standard toleration list. Instance pods inherit this field as is.

- **spec.tolerations.effect**
  string

  Defines the taint's effect this toleration is associated with (it matches any effect if empty).

  Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **spec.tolerations.key**
  string

  Defines the taint's key this toleration is associated with (it matches any key if empty).

  If the key is omitted (emplty), then `operator` must be set to `Exists` (which means it matches any `value` and `key`).

- **spec.tolerations.operator**
  string

  Defines how the key and its value are linked — whether the key must be `Equal` to the value or can have any value.

  Setting it to `Exists` is equivalent to the value having any value so that the pod with the specified toleration can match the corresponding taint.

  Default: `Equal`

  Allowed values: `Exists`, `Equal`

- **spec.tolerations.tolerationSeconds**
  integer

  Specifies the period (in seconds) that the scheduler waits before evicting a Pod from a node if the toleration no longer matches the taint (**only** applicable to the `NoExecute` effect, otherwise ignored).

  If the parameter **is not set**, the Pod will not be evicted from the node if the toleration no longer matches the taint. If the parameter **is set to zero** (or negative), the Pod will be evicted immediately if the toleration no longer matches the taint.

  The default is: not set.

- **spec.tolerations.value**
  string

  The value that the toleration must match.

  Must be empty if the operator parameter is set to `Exists`.

- **spec.vrrpInstances**
  array of objects

  Required value

  A list of VRRP instances in the keepalived cluster. Essentially, it is a list of address groups that migrate between servers simultaneously and cannot work separately.

  Do not confuse `vrrpInstance` with `KeepalivedInstance`. `vrrpInstance` is a component of `KeepalivedInstance`.

  The `keepalived` module configures VRRP instances so that all addresses (all groups) are evenly distributed across all servers (instead of gathering together on the same node).

  - **spec.vrrpInstances.id**
    integer

    Required value

    A unique **cluster-wide** identifier of a VRRP group.

    Do not use the same ID in different `KeepalivedInstance` instances unless you have a perfect reason to do so.

    Allowed values: `1 <= X <= 255`

  - **spec.vrrpInstances.interface**
    object

    Required value

    Defines the strategy of detecting interface for the service VRRP traffic on the node.

    - **spec.vrrpInstances.interface.detectionStrategy**
      string

      You can select one of the following three interface detecting strategis:

      - `Name` — set the interface name explicitly using the `spec.vrrpInstances[].interface.name` parameter. In this case, all nodes must have the same interface that serves the correct network (e.g., eth0).
      - `NetworkAddress` — find an interface with an IP from this subnet on the node and use it.
      - `DefaultRoute` — use the interface for the default route on the node (254 "main" in the table).
    - **spec.vrrpInstances.interface.name**
      string

      The name of the interface for the service VRRP traffic if the `Name` `detectionStrategy` is used.

      A mandatory parameter if `detectionStrategy` = `Name`.

    - **spec.vrrpInstances.interface.networkAddress**
      string

      The address of the subnet to use for determining an interface if `detectionStrategy` = `NetworkAddress`.

      A mandatory parameter if `detectionStrategy` = `NetworkAddress`.

      Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

      Example:

```
networkAddress: 192.168.42.0/24
```

- **spec.vrrpInstances.preempt**
  boolean

  Defines if the IP is passed to a node after its recovery from a failure.

  It makes sense not to pass the IP if you have just one `vrrpInstance` to preserve connections. If you have a lot of groups and a lot of traffic, it is better to pass it to prevent the concentration of all groups on one node.

  The default is — `true`, that is, the IP will passed to a node after its recovery.

- **spec.vrrpInstances.virtualIPAddresses**
  array of objects

  Required value

  A list of IP addresses that will “jump” between the servers **in tandem**

  - **spec.vrrpInstances.virtualIPAddresses.address**
    string

    Required value

    One of the group addresses.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

    Example:

    ```
    address: 192.168.42.15/32
    ```

  - **spec.vrrpInstances.virtualIPAddresses.interface**
    object

    Not required value.

    An interface for binding a virtual IP address on a node (similar to `spec.vrrpInstances[].interface`).

    If omitted, the main service interface defined in `spec.vrrpInstances[].interface` will be used.

    - **spec.vrrpInstances.virtualIPAddresses.interface.detectionStrategy**
      string

      You can select one of the following three interface detecting strategis ti bind an IP address:

      - `Name` — set the interface name explicitly using the `spec.vrrpInstances[].virtualIPAddresses[].interface.name` parameter. In this case, all nodes must have the same interface that serves the correct network (e.g., eth0).
      - `NetworkAddress` — find an interface with an IP from this subnet on the node and use it.
      - `DefaultRoute` — use the interface for the default route on the node (254 “main” in the table).
    - **spec.vrrpInstances.virtualIPAddresses.interface.name**
      string

      A name of the interface to bind a virtual IP address if `virtualIPaddresses[].detectionStrategy = Name` is used.

      A mandatory parameter if `virtualIPaddresses[].detectionStrategy = Name`.

    - **spec.vrrpInstances.virtualIPAddresses.interface.networkAddress**
      string

      The address of the subnet to use for determining an interface if `virtualIPaddresses[].detectionStrategy = NetworkAddress`.

      A mandatory parameter if `virtualIPaddresses[].detectionStrategy = NetworkAddress`.

      Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

      Example:

      ```
      networkAddress: 192.168.42.0/24
      ```

keepalived: FAQ

> **i** The module lifecycle stage: General Availability

How to manually switch keepalived?

1. Enter the desired pod using a debug container with a shared process namespace: `d8 k debug -n d8-keepalived -it keepalived-<name> --profile=general --target keepalived`.
2. Edit the configuration file `vim /proc/1/root/etc/keepalived/keepalived.conf`, replace the value in the `priority` line with <number of keepalived pods + 1> or set a value higher than the current VRRP master (e.g., `255`).
3. Apply settings – send a signal to reload the configuration: `kill -HUP 1`.

Examples

> **i** The module lifecycle stage: General Availability

Three public IP addresses

Suppose there are three public IP addresses on three front nodes. Each virtual IP address is placed in a separate VRRP group. Thus, each

address “jumps” independently of the others, and if there are three nodes in the cluster with the `node-role.deckhouse.io/frontend: ""` labels, then each IP gets its own MASTER node.

```
apiVersion: deckhouse.io/v1alpha1
kind: KeepalivedInstance
metadata:
  name: front
spec:
  nodeSelector: # Mandatory.
    node-role.deckhouse.io/frontend: ""
  tolerations:  # Optional.
  - key: dedicated.deckhouse.io
    operator: Equal
    value: frontend
  vrrpInstances:
  - id: 1 # The unique cluster ID.
    interface:
      detectionStrategy: DefaultRoute # The card with the default route is used as a service network one.
    virtualIPAddresses:
    - address: 42.43.44.101/32
      # The interface parameter is omitted since IP addresses are based on the cards that service VRRP traffic.
  - id: 2
    interface:
      detectionStrategy: DefaultRoute
    virtualIPAddresses:
    - address: 42.43.44.102/32
  - id: 3
    interface:
      detectionStrategy: DefaultRoute
    virtualIPAddresses:
    - address: 42.43.44.103/32
```

Suppose there is a gateway with a pair of IP addresses for LAN and WAN. In the case of the gateway, the private and public IPs are bind together, and they will “jump” between the nodes in tandem. In the below example, the VRRP traffic is routed through the LAN interface. It can be detected using the NetworkAddress method (assuming that each node has an IP belonging to this subnet).

```
apiVersion: deckhouse.io/v1alpha1
kind: KeepalivedInstance
metadata:
  name: mygateway
spec:
  nodeSelector:
    node-role.deckhouse.io/mygateway: ""
  tolerations:
  - key: node-role.deckhouse.io/mygateway
    operator: Exists
  vrrpInstances:
  - id: 4 # Since "1", "2", "3" IDs are used in the "front" KeepalivedInstance above.
    interface:
      detectionStrategy: NetworkAddress
      networkAddress: 192.168.42.0/24
    virtualIPAddresses:
    - address: 192.168.42.1/24
      # In this case, we have already detected the local network (above); thus, the interface parameter can be
safely omitted.
    - address: 42.43.44.1/28
      interface:
        detectionStrategy: Name
        name: ens7 # we use the fact that an interface for public IPs is called "ens7" on all nodes
```

The kube-dns module

> **i** The module lifecycle stage: General Availability

The module installs CoreDNS components for managing DNS in the Kubernetes cluster.

> **i** **Caution.** The module deletes all the previously installed kubeadm Deployments, ConfigMaps as well as RBAC for CoreDNS. When deploying your own CoreDNS, avoid using the names `coredns` or `system:coredns` for any resources (Deployment, Service, ConfigMap, ServiceAccount, ClusterRole, ClusterRoleBinding). Use alternative names like `infra-dns` to prevent automatic removal by Deckhouse Kubernetes Platform.

Third-party components

List of third-party software used in the `kube-dns` module:

- CoreDNS

  License: Apache License 2.0

  A DNS server/forwarder that chains plugins.

Configuration

> The module lifecycle stage: General Availability

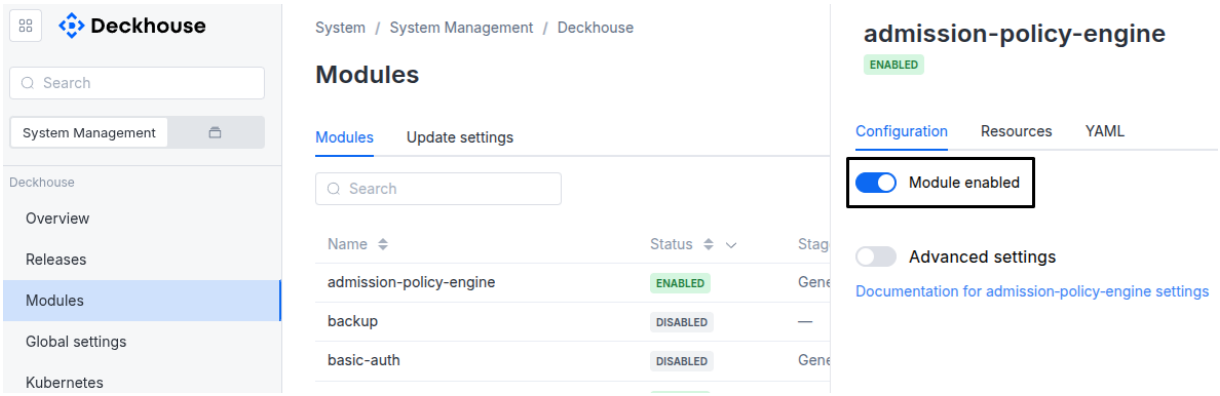
The module has 1 alert.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `kube-dns` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable kube-dns
```

- **Using ModuleConfig `kube-dns`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `kube-dns` (create it if necessary);

  Example of a manifest to enable module `kube-dns`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  enabled: true
```

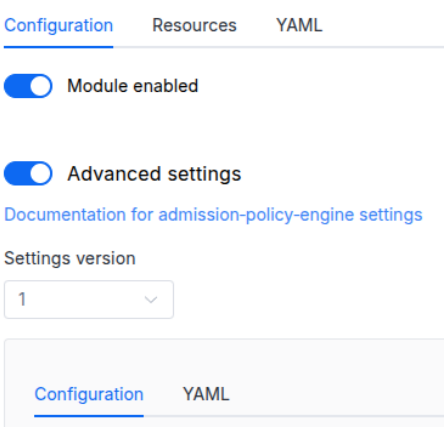
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `kube-dns` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `kube-dns` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `kube-dns`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `kube-dns` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc kube-dns
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `kube-dns` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `kube-dns`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: kube-dns
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.clusterDomainAliases**
    array of strings

    A list of cluster domain aliases; these are resolved on par with `global.discovery.clusterDomain`.

    Please note: the domain alias must not match the domain used in the DNS name template in the publicDomainTemplate parameter.

    - **Element of the array**
      string

      Pattern: `^[0-9a-zA-Z\.-]+$`
  - **settings.enableLogs**
    boolean

    Enable CoreDNS logging.

    Default: `false`
  - **settings.hosts**
    array of objects

    Not required value.

    A static list of hosts similar to that of `/etc/hosts`.

    - **settings.hosts.domain**
      string

      Pattern: `^[0-9a-zA-Z\.-]+$`
    - **settings.hosts.ip**
      string

      Pattern: `^[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}$`
  - **settings.stubZones**
    array of objects

    A list of additional zones CoreDNS should be authoritative for.

    Default: `[]`

    Example:

    ```
    stubZones:
    - zone: example.com
      upstreamNameservers:
      - 8.8.8.8
      cacheTTLSeconds: 3600
      transportProtocolMode: PreferUDP
    ```

    - **settings.stubZones.cacheTTLSeconds**
      integer

      Max TTL in seconds for NOERROR responses.

      Default: `30`

      Allowed values: `1 <= X <= 3600`

- **settings.stubZones.transportProtocolMode**
  string

  The mode of the transport protocol for communicating with the upstream DNS server:

  - `PreferUDP` — UDP is used, even if the original request came over TCP. Note that if the DNS response from the upstream DNS server exceeds 512 bytes and has been truncated and marked with the TC bit (RFC 1035), then it will be resent over TCP (RFC 5966). The request will not be resent over TCP if there are other UDP issues.
  - `ForceTCP` — Always uses TCP, even if the original request is over UDP.
  - `KeepOriginal` — Uses the same protocol (TCP or UDP) as the original request.

  Default: `PreferUDP`

  Allowed values: `PreferUDP`, `ForceTCP`, `KeepOriginal`

- **settings.stubZones.upstreamNameservers**
  array of strings

  Required value

  A list of IP addresses of recursive DNS servers that CoreDNS will use to resolve domains in this zone.

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}(:[0-9]{1,})?$`

- **settings.stubZones.zone**
  string

  Required value

  The CoreDNS zone.

  Pattern: `^[0-9a-zA-Z\.\-_]+$`

  Example:

  ```
  zone: consul.local
  ```

- **settings.transportProtocolMode**
  string

  The mode of the transport protocol for communicating with the upstream DNS server:

  - `PreferUDP` — UDP is used, even if the original request came over TCP. Note that if the DNS response from the upstream DNS server exceeds 512 bytes and has been truncated and marked with the TC bit (RFC 1035), then it will be resent over TCP (RFC 5966). The request will not be resent over TCP if there are other UDP issues.
  - `ForceTCP` — Always uses TCP, even if the original request is over UDP.
  - `KeepOriginal` — Uses the same protocol (TCP or UDP) as the original request.

  Default: `PreferUDP`

  Allowed values: `PreferUDP`, `ForceTCP`, `KeepOriginal`

- **settings.upstreamNameservers**
  array of strings

  A list of IP addresses of recursive DNS servers that CoreDNS will use to resolve external domains.

  By default, the `/etc/resolv.conf` list is used.

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}\.[0-9]{1,}(:[0-9]{1,})?$`

FAQ

> **i** The module lifecycle stage: General Availability

How to change cluster domain with minimal downtime?

Add a new domain and retain the previous one. To do this, modify the configuration parameters:

1. In the controlPlaneManager.apiserver:
   - controlPlaneManager.apiserver.certSANs,
   - apiserver.serviceAccount.additionalAPIAudiences,
   - apiserver.serviceAccount.additionalAPIIssuers.

   Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: control-plane-manager
spec:
  enabled: true
  version: 1
  settings:
    apiserver:
      certSANs:
       - kubernetes.default.svc.<old clusterDomain>
       - kubernetes.default.svc.<new clusterDomain>
      serviceAccount:
        additionalAPIAudiences:
        - https://kubernetes.default.svc.<old clusterDomain>
        - https://kubernetes.default.svc.<new clusterDomain>
        additionalAPIIssuers:
        - https://kubernetes.default.svc.<old clusterDomain>
        - https://kubernetes.default.svc.<new clusterDomain>
```

2. In the kubeDns.clusterDomainAliases:

   Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  version: 1
  enabled: true
  settings:
    clusterDomainAliases:
      - <old clusterDomain>
      - <new clusterDomain>
```

3. Wait until the `kube-apiserver` has restarted.

```
d8 k -n kube-system get pods -l component=kube-apiserver
```

4. Replace the previous `clusterDomain` with the new one. Run this command to edit cluster configuration:

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

5. Restart deckhouse pods:

```
kubectl -n d8-system rollout restart deployment deckhouse
```

⚠ **Important.** In kubernetes, controllers in the cluster use advanced ServiceAccount tokens to interact with API server. These tokens have extra fields `iss:` and `aud:` that contain `clusterDomain` (e.g. `"iss": "https://kubernetes.default.svc.cluster.local"`). After changing the clusterDomain, the API server will start issuing tokens with the new service-account-issuer, but thanks to the configuration of additionalAPIAudiences and additionalAPIIssuers, the apiserver will continue to accept the old tokens. After 48 minutes (80% of 3607 seconds), Kubernetes will begin to refresh the issued tokens, and the new service-account-issuer will be used for the updated tokens. After 90 minutes (3607 seconds plus a short buffer) following the kube-apiserver restart, you can remove the serviceAccount configuration from the control-plane-manager configuration.

⚠ **Important.** In case you need to remove old domain from `clusterDomainAliases` in kube-dns configuration, recreation of all pods in cluster is required, so new pods will have correct search domain in `/etc/resolv.conf` file. This will lead to service downtime until pods are restarted.

```
d8 k delete pods --all-namespaces --all
```

⚠ **Important!** If you use istio module, you have to restart all the application pods under istio control after changing `clusterDomain`.

How to scale-up kube-dns replicas?

Deckhouse distributes kube-dns pods based on the following principles. It searches for nodes with the labels `node-role.deckhouse.io/` and `node-role.kubernetes.io/`, then applies the following rules:

- If there are nodes with the `kube-dns` role in the cluster, the number of replicas is calculated as the sum of these nodes and the master nodes, but not more than masters count plus 2.
- If kube-dns nodes are absent, the system searches for nodes with the `system` role, and the number of replicas is then determined as the sum of system nodes and master nodes, but not more than masters count plus 2.
- If only master nodes are present in the cluster, the number of kube-dns replicas will be equal to the number of masters.

Examples

> The module lifecycle stage: General Availability

Configuration example

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: kube-dns
spec:
  version: 1
  enabled: true
  settings:
    upstreamNameservers:
    - 8.8.8.8
    - 8.8.4.4
    hosts:
    - domain: one.example.com
      ip: 192.168.0.1
    - domain: two.another.example.com
      ip: 10.10.0.128
    stubZones:
    - zone: consul.local
      upstreamNameservers:
      - 10.150.0.1
    enableLogs: true
    clusterDomainAliases:
    - foo.bar
    - baz.qux
```

The kube-proxy module

> The module lifecycle stage: General Availability

The module deletes the entire `kubeadm` kube-proxy set (`DaemonSet`, `ConfigMap`, `RBAC`) and installs its own.

For security reasons, for NodePort services, connections are only allowed to the nodes' InternalIP by default. You can lift this restriction using the `node.deckhouse.io/nodeport-bind-internal-ip: "false"` annotation.

Here is an example of a NodeGroup annotation:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: myng
spec:
  nodeTemplate:
    annotations:
      node.deckhouse.io/nodeport-bind-internal-ip: "false"
...
```

> **Note!** Following the addition, deletion, or changing the annotation, you have to restart kube-proxy Pods manually.
>
> **Note!** The kube-proxy module is automatically disabled when the cni-cilium module is enabled.

The local-path-provisioner module

> The module lifecycle stage: General Availability

Local Path Provisioner provides a way for the Kubernetes users to utilize the local storage on each node.

How does it work?

For each custom resource LocalPathProvisioner, a corresponding `StorageClass` is created.

The allowed topology for `StorageClass` is calculated based on the list of `nodeGroup` names from the CR. The topology is used for scheduling Pods.

When a Pod orders a disk:

- a `HostPath` PV is created
- the `Provisioner` creates a local disk folder on the desired node along the path consisting of the `path` custom resource parameter, the PV name and the PVC name.

  Example of a path:

  ```
  /opt/local-path-provisioner/pvc-d9bd3878-f710-417b-a4b3-38811aa8aac1_d8-monitoring_prometheus-main-db-prometheus-main-0
  ```

Limitations

- The disk size limit is not supported for the local path provisioned volumes.

Third-party components

List of third-party software used in the `local-path-provisioner` module:

- Local-path-provisioner

  License: Apache License 2.0

  Local Path Provisioner provides a way for the Kubernetes users to utilize the local storage in each node by creating hostPath based persistent volume on the node automatically.

Configuration

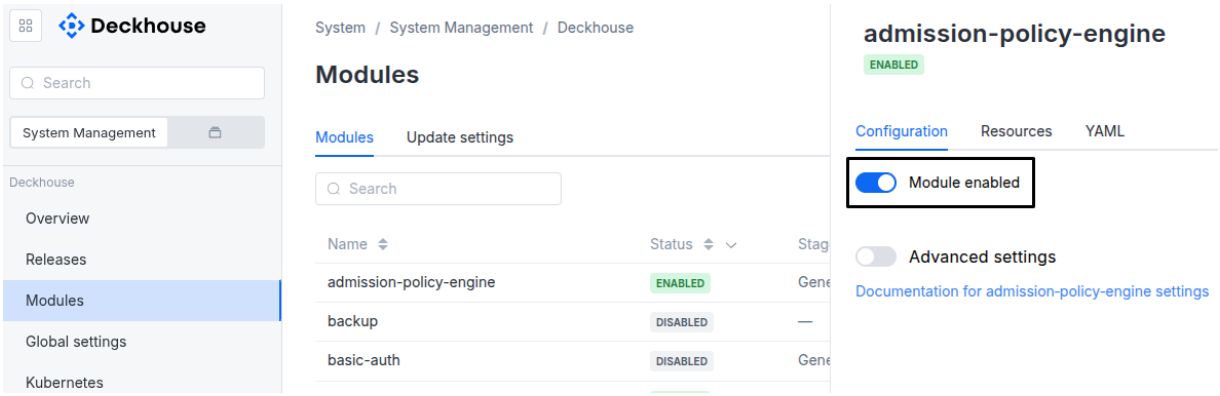
> i The module lifecycle stage: General Availability

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `local-path-provisioner` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable local-path-provisioner
```

- **Using ModuleConfig `local-path-provisioner`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `local-path-provisioner` (create it if necessary);

  Example of a manifest to enable module `local-path-provisioner`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: local-path-provisioner
spec:
  enabled: true
```

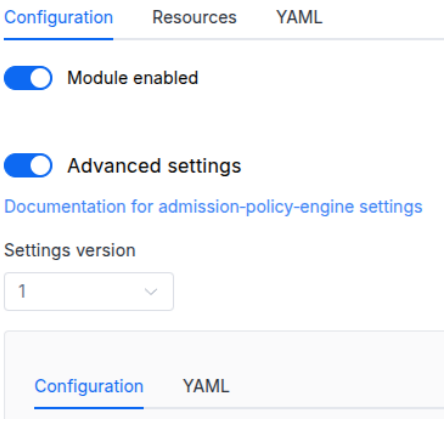
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `local-path-provisioner` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `local-path-provisioner` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `local-path-provisioner`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `local-path-provisioner` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc local-path-provisioner
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `local-path-provisioner` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `local-path-provisioner`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: local-path-provisioner
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

The module does not require any configuration.

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

LocalPathProvisioner

Scope: Cluster
Version: v1alpha1

Parameters of the `local-path-provisioner` instance.

- **spec**
  object
  - **spec.nodeGroups**
    array of strings

    Select nodeGroups for storageClass allowedTopologies.

    Example:

    ```
    nodeGroups:
    - master
    - worker
    ```

  - **spec.path**
    string

    Required value

Node root path for PV's.

Example:

```
path: "/opt/local-path-provisioner"
```

- **spec.reclaimPolicy**
  string

  Retain policy for PV's.

  Default: `Retain`

  Allowed values: `Retain`, `Delete`

FAQ

> **i** The module lifecycle stage: General Availability

How to configure Prometheus to use local storage for storing data?

Deploy custom resource `LocalPathProvisioner`:

```
apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
spec:
  nodeGroups:
  - system
  path: "/opt/local-path-provisioner"
```

- `spec.nodeGroups` must match NodeGroup where prometheus Pods run.
- `spec.path` - node data path.

Add to the following parameters to the `prometheus` module configuration:

```
longtermStorageClass: localpath-system
storageClass: localpath-system
```

Wait for the restart of Prometheus Pods.

Examples

> **i** The module lifecycle stage: General Availability

Example of a `LocalPathProvisioner` custom resource

Reclaim policy set by default to `Retain`.

```
apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
spec:
  nodeGroups:
  - system
  path: "/opt/local-path-provisioner"
```

Example of a `LocalPathProvisioner` custom resource with `reclaimPolicy` set

Reclaim policy set to `Delete`.

```
apiVersion: deckhouse.io/v1alpha1
kind: LocalPathProvisioner
metadata:
  name: localpath-system
spec:
  nodeGroups:
  - system
  path: "/opt/local-path-provisioner"
  reclaimPolicy: "Delete"
```

The log-shipper module

> **i** The module lifecycle stage: General Availability

The `log-shipper` module simplifies the configuration of log collection in Kubernetes. It allows you to quickly collect logs from both applications running in the cluster and from the nodes themselves, and then send them to any storage system, internal or external (for

example, Loki, Elasticsearch, S3, and others).

Module features:

- centrally collects and transmits logs from the cluster;
- filters, transforms, and enriches logs before sending;
- configures routing of logs between different sources and receivers.

1. Deckhouse Kubernetes Platform (DKP) is watching ClusterLoggingConfig, ClusterLogDestination and PodLoggingConfig custom resources. The combination of a logging source and log destination is called `pipeline`.
2. DKP generates a configuration file and stores it into Kubernetes `Secret`.
3. `Secret` is mounted to all log-shipper agent Pods and the configuration is reloaded on changes by the `reloader` sidecar container.

Deployment topologies

This module deploys only agents on nodes. It is implied that logs are sent from the cluster using one of the following topologies.

Distributed

Agents send logs directly to the storage, e.g., Loki, Elasticsearch.

- Less complicated scheme to use.
- Available out of the box without any external dependency besides storage.
- Complicated transformations consume more resources.

Centralized

All logs are aggregated by one of the available aggregation destinations, e.g., Logstash, Vector. Agents on nodes do minimal transformations and try to send logs from nodes faster with less resource consumption. Complicated mappings are applied on the aggregator’s side.

- Lower resource consumption for applications on nodes.
- Users can configure any possible mappings for aggregators and send logs to many more storages.
- Dedicated nodes for aggregates can be scaled up and down on the loading changes.

Stream

The main goal of this architecture is to send messages to the queue system as quickly as possible, then other workers will read them and deliver them to the long-term storage for later analysis.

- The same pros and cons as for centralized architecture, yet one more middle layer storage is added.
- Increased durability. Suites for all infrastructures where logs delivery is crucial.

Metadata

On collecting, all sources enrich logs with metadata. The enrichment takes place at the `Source` stage.

Kubernetes

The following metadata fields will be exposed:

Label	Pod spec path
`pod`	metadata.name
`namespace`	metadata.namespace
`pod_labels`	metadata.labels
`pod_ip`	status.podIP
`image`	spec.containers[].image
`container`	spec.containers[].name
`node`	spec.nodeName
`pod_owner`	metadata.ownerRef[0]
Label	**Node spec path**
`node_group`	metadata.labels[].node.deckhouse.io/group

i For Splunk, the `pod_labels` fields are not exported because it is a nested object, which is not supported by Splunk.

File

The following metadata fields will be exposed:

Label	Pod spec path
`host`	spec.nodeName
`host_ip`	status.hostIP
`file`	Path to the log file being read

Log filters

There are two filters for reducing the number of messages sent to storage, `log filter` and `label filter`.

They are executed right after concatenating lines together with the multiline log parser.

1. `label filter` - rules are run on message metadata. The metadata (or label) fields are populated based on the log source, and different sources will have different sets of fields. These rules are needed, for example, to discard messages from a specific container or pod with/without a label.
2. `log filter` - rules are run on the original message. It is possible to discard a message based on a JSON field, or, if the message is not in JSON format, use a regular expression to search the string.

Both filters have the same structured configuration:

- `field` — the source of data to filter (most of the time it is a value of a label or a JSON parameter).
- `operator` — action to apply to a value of the field. Possible options are In, NotIn, Regex, NotRegex, Exists, DoesNotExist.
- `values` — defines different values for different operators:
  - DoesNotExist, Exists — not supported;
  - In, NotIn — a value of a field must / mustn't be in the list of provided values;
  - Regex, NotRegex — a value of a field must match any or mustn't match all the provided regexes (values).

More examples can be found in the Examples section of the documentation.

⚠ Extra labels are added on the `Destination` stage of the pipeline, so it is impossible to run queries against them.

Third-party components

List of third-party software used in the `log-shipper` module:

- Vector

  License: Mozilla Public License, version 2.0

  Vector is a high-performance, observability data pipeline for logs and metrics

Configuration

i The module lifecycle stage: General Availability

The module starts reading logs only if log-pipeline is created. Log-pipeline consists of ClusterLoggingConfig/PodLoggingConfig connected to ClusterLogDestination.

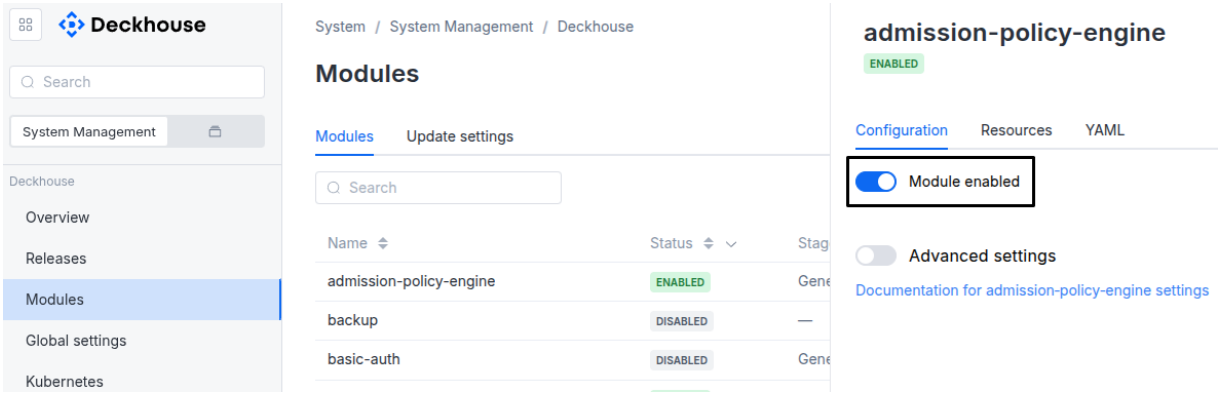
The module has 6 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `log-shipper` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable log-shipper
```

- **Using ModuleConfig `log-shipper`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `log-shipper` (create it if necessary);

  Example of a manifest to enable module `log-shipper`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: log-shipper
spec:
  enabled: true
```

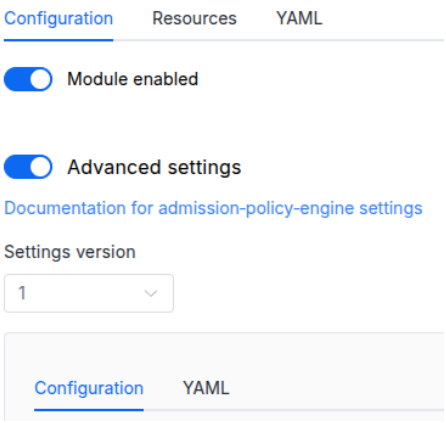
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `log-shipper` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `log-shipper` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `log-shipper`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `log-shipper` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc log-shipper
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `log-shipper` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `log-shipper`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: log-shipper
  spec:
    version: 2
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Move `.settings.resourcesRequests.static.cpu` and `.settings.resourcesRequests.static.memory` to `.settings.resourcesRequests.static.requests`.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.debug**
    boolean

Enable or disable agent debug logging. Default: `false`.

Default: `false`

- **settings.nodeSelector**
  object

  The same as in the Pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, `nodeSelector` will be determined automatically.

- **settings.resourcesRequests**
  object

  Max amounts of CPU and memory resources that the pod can request when selecting a node.

  If the `vertical-pod-autoscaler` module is disabled, then these values become the default ones.

  Examples:

```
resourcesRequests:
  mode: VPA
  vpa:
    mode: Auto
    cpu:
      min: 50m
      max: 2
    memory:
      min: 256Mi
      max: 2Gi
```

```
resourcesRequests:
  mode: VPA
  vpa:
    mode: Initial
    cpu:
      min: 1
      max: 3000m
    memory:
      min: 1024
      max: 4096
```

```
resourcesRequests:
  mode: Static
  static:
    requests:
      cpu: 55m
      memory: 256Mi
    limits:
      cpu: 2
      memory: 2Gi
```

  - **settings.resourcesRequests.mode**
    string

    Required value

    Resource request management mode:

    - `Static` is a classic one. In it, you explicitly specify requests. The parameters of this mode are defined in the static parameter section;
    - `VPA` mode uses VPA. You can configure this mode by modifying parameters in the vpa parameter section.

    Default: `VPA`

    Allowed values: `VPA`, `Static`

  - **settings.resourcesRequests.static**
    object

    Resource request management options for the `Static` mode.

    - **settings.resourcesRequests.static.limits**
      object

      Configuring CPU and memory limits.

      - **settings.resourcesRequests.static.limits.cpu**
        string or number

        Configuring CPU limits.

        Pattern: `^[0-9]+m?$`

      - **settings.resourcesRequests.static.limits.memory**
        string or number

        Configuring memory limits.

        Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    - **settings.resourcesRequests.static.requests**
      object

Resource requests settings for pods.

- **settings.resourcesRequests.static.requests.cpu**
  string or number

  Configuring CPU requests.

  Pattern: `^[0-9]+m?$`

- **settings.resourcesRequests.static.requests.memory**
  string or number

  Configuring memory requests.

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesRequests.vpa**
  object

  Resource request management options for the VPA mode.

  - **settings.resourcesRequests.vpa.cpu**
    object

    Required value

    CPU-related parameters.

    - **settings.resourcesRequests.vpa.cpu.max**
      string or number

      The maximum value that the VPA can set for the CPU requests.

      Default: `500m`

      Pattern: `^[0-9]+m?$`

    - **settings.resourcesRequests.vpa.cpu.min**
      string or number

      The minimum value that the VPA can set for the CPU requests.

      Default: `50m`

      Pattern: `^[0-9]+m?$`

  - **settings.resourcesRequests.vpa.memory**
    object

    Required value

    The amount of memory requested.

    - **settings.resourcesRequests.vpa.memory.max**
      string or number

      The maximum memory requests the VPA can set.

      Default: `2048Mi`

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    - **settings.resourcesRequests.vpa.memory.min**
      string or number

      The minimum memory requests the VPA can set.

      Default: `64Mi`

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.resourcesRequests.vpa.mode**
    string

    Required value

    The VPA usage mode.

    Default: `Initial`

    Allowed values: `Initial`, `Auto`

- **settings.tolerations**
  array of objects

  The same as in the Pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, `tolerations` will be determined automatically.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

ClusterLogDestination

Scope: Cluster

- v1alpha2
- v1alpha1

Describes setting for a log storage, which you can use in many log sources.

`metadata.name` — is an upstream name, which you should use in custom resource ClusterLoggingConfig.

- **spec**
  object

  Required value

  - **spec.buffer**
    object

    Buffer parameters.

    - **spec.buffer.disk**
      object

      Disk buffer parameters.

      - **spec.buffer.disk.maxSize**
        integer or string

        The maximum size of the buffer on disk. Must be at least ~256MB (268435488 bytes).

        You can express size as a plain integer or as a fixed-point number using one of these quantity suffixes: `E`, `P`, `T`, `G`, `M`, `k`, `Ei`, `Pi`, `Ti`, `Gi`, `Mi`, `Ki`.

        More about resource quantity:

        - kubernetes quantity
        - memory resource units

        Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

        Examples:

        ```
        maxSize: 512Mi
        ```

        ```
        maxSize: 268435488
        ```

    - **spec.buffer.memory**
      object
      - **spec.buffer.memory.maxEvents**
        number

        The maximum number of events allowed in the buffer.

    - **spec.buffer.type**
      string

      Required value

      The type of buffer to use.

      Allowed values: `Disk`, `Memory`

    - **spec.buffer.whenFull**
      string

      Event handling behavior when a buffer is full.

      Default: `Block`

      Allowed values: `DropNewest`, `Block`

  - **spec.elasticsearch**
    object
    - **spec.elasticsearch.auth**
      object
      - **spec.elasticsearch.auth.awsAccessKey**

string

Base64-encoded AWS `ACCESS_KEY`.

- **spec.elasticsearch.auth.awsAssumeRole**
  string

  The ARN of an IAM role to assume at startup.

- **spec.elasticsearch.auth.awsRegion**
  string

  AWS region for authentication.

- **spec.elasticsearch.auth.awsSecretKey**
  string

  Base64-encoded AWS `SECRET_KEY`.

- **spec.elasticsearch.auth.password**
  string

  Base64-encoded Basic authentication password.

- **spec.elasticsearch.auth.strategy**
  string

  The authentication strategy to use.

  Default: `Basic`

  Allowed values: `Basic`, `AWS`

- **spec.elasticsearch.auth.user**
  string

  The Basic authentication user name.

- **spec.elasticsearch.dataStreamEnabled**
  boolean

  Use for storage indexes or datastreams (https://www.elastic.co/guide/en/elasticsearch/reference/master/data-streams.html).

  Datastream usage is better for logs and metrics storage but they works only for Elasticsearch >= 7.16.X.

  Default: `false`

- **spec.elasticsearch.docType**
  string

  The `doc_type` for your index data. This is only relevant for Elasticsearch <= 6.X.

  - For Elasticsearch >= 7.X you do not need this option since this version has removed `doc_type` mapping;
  - For Elasticsearch >= 6.X the recommended value is `_doc`, because using it will make it easy to upgrade to 7.X;
  - For Elasticsearch < 6.X you can't use a value starting with `_` or empty string. Use, for example, values like `logs`.

- **spec.elasticsearch.endpoint**
  string

  Required value

  Base URL of the Elasticsearch instance.

- **spec.elasticsearch.index**
  string

  Index name to write events to.

- **spec.elasticsearch.pipeline**
  string

  Name of the pipeline to apply.

- **spec.elasticsearch.tls**
  object

  Configures the TLS options for outgoing connections.

  - **spec.elasticsearch.tls.caFile**
    string

    Base64-encoded CA certificate in PEM format.

  - **spec.elasticsearch.tls.clientCrt**
    object

    Configures the client certificate for outgoing connections.

    - **spec.elasticsearch.tls.clientCrt.crtFile**
      string

Required value

Base64-encoded certificate in PEM format.

You must also set the `keyFile` parameter.

- **spec.elasticsearch.tls.clientCrt.keyFile**
  string

  Required value

  Base64-encoded private key in PEM format (PKCS#8).

  You must also set the `crtFile` parameter.

- **spec.elasticsearch.tls.clientCrt.keyPass**
  string

  Base64-encoded pass phrase used to unlock the encrypted key file.

- **spec.elasticsearch.tls.secretRef**
  object

  Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

  - **spec.elasticsearch.tls.secretRef.name**
    string

    Name of the Secret with TLS certificates.

- **spec.elasticsearch.tls.verifyCertificate**
  boolean

  Validate the TLS certificate of the remote host. Specifically the issuer is checked but not CRLs (Certificate Revocation Lists).

  Default: `true`

- **spec.elasticsearch.tls.verifyHostname**
  boolean

  Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

  Default: `true`

- **spec.extraLabels**
  object

  A set of labels that will be attached to each batch of events.

  You can use simple templating here: `{{ app }}`.

  There are some reserved keys:

  - parsed_data
  - pod
  - pod_labels_*
  - pod_ip
  - namespace
  - image
  - container
  - node
  - pod_owner

  More about field path notation…

  Example:

  ```
  extraLabels:
    forwarder: vector
    key: value
    app_info: "{{ app }}"
    array_member: "{{ array[0] }}"
    symbol_escating_value: "{{ pay\\.day }}"
  ```

- **spec.kafka**
  object
  - **spec.kafka.bootstrapServers**
    array of strings

    Required value

    A list of host and port pairs that are the addresses of the Kafka brokers in a "bootstrap" Kafka cluster that a Kafka client connects to initially to bootstrap itself.

    Default: `[]`

Example:

```
bootstrapServers:
- 10.14.22.123:9092
- 10.14.23.332:9092
```

- **Element of the array**
  string

  Pattern: `^(.+)\:\d{1,5}$`

- **spec.kafka.encoding**
  object

  How to encode the message.

  - **spec.kafka.encoding.cef**
    object

    CEF-specific configuration fields. Only applicable when codec is set to `CEF`.

    - **spec.kafka.encoding.cef.deviceProduct**
      string

      Default: `log-shipper-agent`

    - **spec.kafka.encoding.cef.deviceVendor**
      string

      Default: `Deckhouse`

    - **spec.kafka.encoding.cef.deviceVersion**
      string

      Default: `1`

  - **spec.kafka.encoding.codec**
    string

    Available encoding formats.

    Default: `JSON`

    Allowed values: `JSON`, `CEF`

- **spec.kafka.keyField**
  string

  Allows to set the key_field.

  Examples:

  ```
  keyField: host
  ```

  ```
  keyField: node
  ```

  ```
  keyField: namespace
  ```

  ```
  keyField: parsed_data.app_info
  ```

- **spec.kafka.sasl**
  object

  Configuration for SASL authentication when interacting with Kafka.

  - **spec.kafka.sasl.mechanism**
    string

    Required value

    The SASL mechanism to use. Only PLAIN and SCRAM-based mechanisms are supported.

    Allowed values: `PLAIN`, `SCRAM-SHA-256`, `SCRAM-SHA-512`

  - **spec.kafka.sasl.password**
    string

    Required value

    The SASL password.

    Example:

    ```
    password: qwerty
    ```

  - **spec.kafka.sasl.username**
    string

Required value

The SASL username.

Example:

```
username: username
```

- **spec.kafka.tls**
  object

  Configures the TLS options for outgoing connections.

  - **spec.kafka.tls.caFile**
    string

    Base64-encoded CA certificate in PEM format.

  - **spec.kafka.tls.clientCrt**
    object

    Configures the client certificate for outgoing connections.

    - **spec.kafka.tls.clientCrt.crtFile**
      string

      Required value

      Base64-encoded certificate in PEM format.

      You must also set the `keyFile` parameter.

    - **spec.kafka.tls.clientCrt.keyFile**
      string

      Required value

      Base64-encoded private key in PEM format (PKCS#8).

      You must also set the `crtFile` parameter.

    - **spec.kafka.tls.clientCrt.keyPass**
      string

      Base64-encoded pass phrase used to unlock the encrypted key file.

  - **spec.kafka.tls.secretRef**
    object

    Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

    - **spec.kafka.tls.secretRef.name**
      string

      Name of the Secret with TLS certificates.

  - **spec.kafka.tls.verifyCertificate**
    boolean

    Validate the TLS certificate of the remote host.

    Default: `true`

  - **spec.kafka.tls.verifyHostname**
    boolean

    Verifies that the name of the remote host matches the name specified in the remote host’s TLS certificate.

    Default: `true`

- **spec.kafka.topic**
  string

  Required value

  The Kafka topic name to write events to. This parameter supports template syntax, which enables you to use dynamic per-event values.

  Examples:

  ```
  topic: logs
  ```

  ```
  topic: logs-{{unit}}-%Y-%m-%d
  ```

- **spec.logstash**
  object
  - **spec.logstash.endpoint**

string

Required value

Base URL of the Logstash instance.

- **spec.logstash.tls**
  object

  Configures the TLS options for outgoing connections.

  - **spec.logstash.tls.caFile**
    string

    Base64-encoded CA certificate in PEM format.

  - **spec.logstash.tls.clientCrt**
    object

    Configures the client certificate for outgoing connections.

    - **spec.logstash.tls.clientCrt.crtFile**
      string

      Required value

      Base64-encoded certificate in PEM format.

      You must also set the `keyFile` parameter.

    - **spec.logstash.tls.clientCrt.keyFile**
      string

      Required value

      Base64-encoded private key in PEM format (PKCS#8).

      You must also set the `crtFile` parameter.

    - **spec.logstash.tls.clientCrt.keyPass**
      string

      Base64-encoded pass phrase used to unlock the encrypted key file.

  - **spec.logstash.tls.secretRef**
    object

    Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

    - **spec.logstash.tls.secretRef.name**
      string

      Name of the Secret with TLS certificates.

  - **spec.logstash.tls.verifyCertificate**
    boolean

    Validate the TLS certificate of the remote host.

    Default: `true`

  - **spec.logstash.tls.verifyHostname**
    boolean

    Verifies that the name of the remote host matches the name specified in the remote host’s TLS certificate.

    Default: `true`

- **spec.loki**
  object
  - **spec.loki.auth**
    object
    - **spec.loki.auth.password**
      string

      Base64-encoded Basic authentication password.

    - **spec.loki.auth.strategy**
      string

      The authentication strategy to use.

      Default: `Basic`

      Allowed values: `Basic`, `Bearer`

    - **spec.loki.auth.token**
      string

      The token to use for Bearer authentication.

- **spec.loki.auth.user**
  string

  The Basic authentication user name.

- **spec.loki.endpoint**
  string

  Required value

  Base URL of the Loki instance.

  > **i** Agent automatically adds `/loki/api/v1/push` into URL during data transmission.

- **spec.loki.tenantID**
  string

  ID of a tenant.

  This option is used only for GrafanaCloud. When running Loki locally, a tenant ID is not required.

- **spec.loki.tls**
  object

  Configures the TLS options for outgoing connections.

  - **spec.loki.tls.caFile**
    string

    Base64-encoded CA certificate in PEM format.

  - **spec.loki.tls.clientCrt**
    object

    Configures the client certificate for outgoing connections.

    - **spec.loki.tls.clientCrt.crtFile**
      string

      Required value

      Base64-encoded certificate in PEM format.

      You must also set the `keyFile` parameter.

    - **spec.loki.tls.clientCrt.keyFile**
      string

      Required value

      Base64-encoded private key in PEM format (PKCS#8).

      You must also set the `crtFile` parameter.

    - **spec.loki.tls.clientCrt.keyPass**
      string

      Base64-encoded pass phrase used to unlock the encrypted key file.

  - **spec.loki.tls.secretRef**
    object

    Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

    - **spec.loki.tls.secretRef.name**
      string

      Name of the Secret with TLS certificates.

  - **spec.loki.tls.verifyCertificate**
    boolean

    Validate the TLS certificate of the remote host.

    If set to `false`, the certificate is not checked in the Certificate Revocation Lists.

    Default: `true`

  - **spec.loki.tls.verifyHostname**
    boolean

    Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

    Default: `true`

- **spec.rateLimit**
  object

Parameter for limiting the flow of events.

- **spec.rateLimit.excludes**
  array of objects

  List of excludes for keyField.

  Only NOT matched log entries would be rate limited.

  Examples:

  ```
  excludes:
    field: tier
    operator: Exists
  ```

  ```
  excludes:
    field: foo
    operator: NotIn
    values:
    - dev
    - 42
    - 'true'
    - '3.14'
  ```

  ```
  excludes:
    field: bar
    operator: Regex
    values:
    - "^abc"
    - "^\\d.+$"
  ```

  - **spec.rateLimit.excludes.field**
    string

    Required value

    Field name for filtering.

  - **spec.rateLimit.excludes.operator**
    string

    Required value

    Operator for log field comparations:

    - `In` — finds a substring in a string.
    - `NotIn` — is a negative version of the `In` operator.
    - `Regex` — is trying to match regexp over the field; only log events with matching fields will pass.
    - `NotRegex` — is a negative version of the `Regex` operator; log events without fields or with not matched fields will pass.
    - `Exists` — drops log event if it contains some fields.
    - `DoesNotExist` — drops log event if it does not contain some fields.

    Allowed values: `In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`

  - **spec.rateLimit.excludes.values**
    array

    Array of values or regexes for corresponding operations. Does not work for `Exists` and `DoesNotExist` operations.

    Fields a with float or boolean values will be converted to strings during comparison.

- **spec.rateLimit.keyField**
  string

  The name of the log field whose value will be hashed to determine if the event should be rate limited.

- **spec.rateLimit.linesPerMinute**
  number

  Required value

  The number of records per minute.

- **spec.socket**
  object
  - **spec.socket.address**
    string

    Required value

    Address of the socket.

    Pattern: `^.*:[1-9][0-9]+$`

  - **spec.socket.encoding**

object

How to encode the message.

- **spec.socket.encoding.cef**
  object

  CEF-specific configuration fields. Only applicable when codec is set to `CEF`.

  - **spec.socket.encoding.cef.deviceProduct**
    string

    Default: `log-shipper-agent`

  - **spec.socket.encoding.cef.deviceVendor**
    string

    Default: `Deckhouse`

  - **spec.socket.encoding.cef.deviceVersion**
    string

    Default: `1`

- **spec.socket.encoding.codec**
  string

  Available encoding formats.

  Default: `JSON`

  Allowed values: `Text`, `JSON`, `Syslog`, `CEF`, `GELF`

- **spec.socket.mode**
  string

  Required value

  Allowed values: `TCP`, `UDP`

- **spec.socket.tcp**
  object
  - **spec.socket.tcp.tls**
    object

    Configures the TLS options for outgoing connections.

    - **spec.socket.tcp.tls.caFile**
      string

      Base64-encoded CA certificate in PEM format.

    - **spec.socket.tcp.tls.clientCrt**
      object

      Configures the client certificate for outgoing connections.

      - **spec.socket.tcp.tls.clientCrt.crtFile**
        string

        Required value

        Base64-encoded certificate in PEM format.

        You must also set the `keyFile` parameter.

      - **spec.socket.tcp.tls.clientCrt.keyFile**
        string

        Required value

        Base64-encoded private key in PEM format (PKCS#8).

        You must also set the `crtFile` parameter.

      - **spec.socket.tcp.tls.clientCrt.keyPass**
        string

        Base64-encoded pass phrase used to unlock the encrypted key file.

    - **spec.socket.tcp.tls.secretRef**
      object

      Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

      - **spec.socket.tcp.tls.secretRef.name**
        string

        Name of the Secret with TLS certificates.

  - **spec.socket.tcp.verifyCertificate**

boolean

Validate the TLS certificate of the remote host.

If set to `false`, the certificate is not checked in the Certificate Revocation Lists.

Default: `true`

- **spec.socket.tcp.verifyHostname**
  boolean

  Verifies that the name of the remote host matches the name specified in the remote host’s TLS certificate.

  Default: `true`

- **spec.splunk**
  object
  - **spec.splunk.endpoint**
    string

    Required value

    Base URL of the Splunk instance.

    Example:

    ```
    endpoint: https://http-inputs-hec.splunkcloud.com
    ```

  - **spec.splunk.index**
    string

    Index name to write events to.

  - **spec.splunk.tls**
    object

    Configures the TLS options for outgoing connections.

    - **spec.splunk.tls.caFile**
      string

      Base64-encoded CA certificate in PEM format.

    - **spec.splunk.tls.clientCrt**
      object

      Configures the client certificate for outgoing connections.

      - **spec.splunk.tls.clientCrt.crtFile**
        string

        Required value

        Base64-encoded certificate in PEM format.

        You must also set the `keyFile` parameter.

      - **spec.splunk.tls.clientCrt.keyFile**
        string

        Required value

        Base64-encoded private key in PEM format (PKCS#8).

        You must also set the `crtFile` parameter.

      - **spec.splunk.tls.clientCrt.keyPass**
        string

        Base64-encoded pass phrase used to unlock the encrypted key file.

    - **spec.splunk.tls.secretRef**
      object

      Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

      - **spec.splunk.tls.secretRef.name**
        string

        Name of the Secret with TLS certificates.

    - **spec.splunk.tls.verifyCertificate**
      boolean

      Validate the TLS certificate of the remote host.

      Default: `true`

    - **spec.splunk.tls.verifyHostname**

boolean

Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

Default: `true`

- **spec.splunk.token**
  string

  Required value

  Default Splunk HEC token. If an event has a token set in its metadata, it will have priority over the one set here.

- **spec.transformations**
  array of objects

  Ordered list of transformations applied to each log event before it is sent to the destination. Items are executed in array order.

  `AddLabels` runs after `extraLabels` when both are configured.

  Available actions:

  - `ReplaceKeys` — recursively replace matches of the `source` pattern with `target` in the given paths.
  - `ParseMessage` — parse the `message` field using `sourceFormat` and write the result to `targetLabel` (default `.message`; the lone `.` merges into the event root). On parse failure the original `message` is left unchanged.
  - `DropLabels` — remove the value at each `label` path, or if `keepKeys` is set on an item, keep only those child key names inside the object at `label`.
  - `AddLabels` — add labels (literals, `{{ .path }}` templates, optional `when`). Runs after `extraLabels`; on key name conflicts `AddLabels` wins.
  - `ReplaceValue` — replace substrings in scalar string fields using the `source` regex; `target` may reference named groups as `{{ group_name }}`.

  Examples:

  ```
  transformations:
    action: ReplaceKeys
    replaceKeys:
      source: "."
      target: _
      labels:
      - ".pod_labels"
  ```

  ```
  transformations:
    action: ParseMessage
    parseMessage:
      sourceFormat: String
      string:
        regex: "^(?P<msg>.*)$"
        setLabels:
          msg: "{{ msg }}"
  ```

  ```
  transformations:
    action: ParseMessage
    parseMessage:
      sourceFormat: Klog
  ```

  ```
  transformations:
    action: ParseMessage
    parseMessage:
      sourceFormat: JSON
      json:
        depth: 1
  ```

  ```
  transformations:
    action: DropLabels
    dropLabels:
      labels:
      - label: ".first"
      - label: ".second"
      - label: ".pod_labels"
        keepKeys:
        - app
        - team
  ```

```
transformations:
  action: AddLabels
  addLabels:
    when:
    - .namespace == "production"
    setLabels:
      ".env": prod
      ".source_app": "{{ .pod_labels.app }}"
```

```
transformations:
  action: ReplaceValue
  replaceValue:
    source: "\\d+"
    target: REDACTED
    labels:
    - ".message"
```

- **spec.transformations.action**
  string

  Transformation type; selects which parameter block is used.

  Allowed values: `ReplaceKeys`, `ParseMessage`, `DropLabels`, `AddLabels`, `ReplaceValue`

- **spec.transformations.addLabels**
  object

  Adds fields to the event from `setLabels` (literals and `{{ .path }}` templates). Runs after `extraLabels` when both are set; on path conflicts values from this transform win. Optional `when` filter.

  - **spec.transformations.addLabels.setLabels**
    object

    Required value

    Keys are destination label paths in Vector notation: each key starts with `.` relative to the event root (e.g. `.env`, `.source_app`). Values are literals or `{{ .path }}` templates where `path` in the template is a source field path with a leading `.`.

    Examples:

    ```
    setLabels:
      ".tier": backend
    ```

    ```
    setLabels:
      ".source_app": "{{ .pod_labels.app }}"
    ```

  - **spec.transformations.addLabels.when**
    array of strings

    Optional list of conditions combined with logical AND. In comparisons, the field path is a leading dot and dot-separated segments (e.g. `.namespace`, `.pod_labels.app`). Each item is either: a comparison `path ==|!=|=~|!=~ value` (regular expression for `=~` / `!=~`), a `path` with no operator or right-hand side (field exists), or `!.path` (field does not exist).

    Examples:

    ```
    when: .namespace == "production"
    ```

    ```
    when: .pod_labels.app =~ "^api-"
    ```

    ```
    when: .level != "debug"
    ```

    ```
    when: ".pod_labels.team"
    ```

    ```
    when: "!.pod_labels.legacy"
    ```

    - **Element of the array**
      string

      Pattern: `^[a-zA-Z0-9_\s\\\[\](){}?*+.^$\-=!:,'"|<>~#/@]+$`

- **spec.transformations.dropLabels**
  object

  Removes field paths, or narrows objects by keeping only selected child keys at each `label` when `keepKeys` is set.

  - **spec.transformations.dropLabels.labels**
    array of objects

Required value

Each item has a required Vector path `label` and optional `keepKeys`. If `keepKeys` is omitted, the subtree at `label` is removed. If `keepKeys` is set, only those child key names are kept in the object at `label`.

- **spec.transformations.dropLabels.labels.keepKeys**
  array of strings

  Child key names (no leading dot) to keep inside the object at `label`; all other child keys are removed.

  - **Element of the array**
    string

    Pattern: `^[a-zA-Z0-9_.\-/]+$`

- **spec.transformations.dropLabels.labels.label**
  string

  Required value

  Vector path to a field or subtree; when `keepKeys` is set, must resolve to an object.

  Pattern: `^\.[a-zA-Z0-9_\[\]\\\.\-"/:]+$`

- **spec.transformations.parseMessage**
  object

  Parses the string in the `message` field (format from `sourceFormat`) and writes the result to `targetLabel` as a Vector path with a leading `.` (default `.message`; the lone `.` merges the parsed object into the event root). On parse failure the original `message` field is left unchanged.

  - **spec.transformations.parseMessage.json**
    object
    - **spec.transformations.parseMessage.json.depth**
      integer

      Depth for JSON parsing.

      Allowed values: `1 <= X <= 128`

  - **spec.transformations.parseMessage.sourceFormat**
    string

    Required value

    Message format for converting into an object.

    Allowed values: `String`, `JSON`, `Klog`, `SysLog`, `CLF`, `Logfmt`

  - **spec.transformations.parseMessage.string**
    object

    Settings for `sourceFormat: String`. `regex` and `setLabels` are required.

    - **spec.transformations.parseMessage.string.regex**
      string

      Required value

      Regular expression with named capture groups; requires `setLabels`.

      Pattern: `^[a-zA-Z0-9_\s\\\[\](){}?*+.^$\-=!:,'\"|<>~#/@]+$`

    - **spec.transformations.parseMessage.string.setLabels**
      object

      Required value

      Maps output keys to literals or `{{ group_name }}` templates.

  - **spec.transformations.parseMessage.targetLabel**
    string

    Destination path for the parsed value (default `.message`). The lone value `.` merges into the event root. Must not be `.parsed_data`.

    Default: `.message`

    Pattern: `^(\.|\.[a-zA-Z0-9_\[\]\\\.\-"/:']+)$`

    Examples:

    ```
    targetLabel: ".message"
    ```

    ```
    targetLabel: "."
    ```

    ```
    targetLabel: ".tmp"
    ```

- **spec.transformations.replaceKeys**

object

Recursively replaces matches of the `source` pattern with the `target` value in the listed paths (`labels`). Paths use Vector notation with a leading `.` on each list item.

- **spec.transformations.replaceKeys.labels**
  array of strings

  Required value

  Label paths where the replacement runs. Each item must start with `.` (Vector field path notation, e.g. `.pod_labels`, `.parsed_data`).

  Examples:

  ```
  labels: ".pod_labels"
  ```

  ```
  labels: ".annotations"
  ```

  - **Element of the array**
    string

    Pattern: `^\.[a-zA-Z0-9_\[\]\\\.\-"/:']+$`

- **spec.transformations.replaceKeys.source**
  string

  Required value

  Pattern used to find matches. Can be a static string or a regular expression (same character set as other regex fields in transformations).

  Pattern: `^[a-zA-Z0-9_\s\\\[\](){}?*+.^$\-=!:,'"|<>~#/@]+$`

  Examples:

  ```
  source: "."
  ```

  ```
  source: "^old\\."
  ```

- **spec.transformations.replaceKeys.target**
  string

  Literal replacement string for matched substrings (not a regex).

  Default:

  Examples:

  ```
  target: _
  ```

  ```
  target: ''
  ```

- **spec.transformations.replaceValue**
  object

  Replace matches of `source` in each listed scalar string field with `target`.

  - **spec.transformations.replaceValue.labels**
    array of strings

    Required value

    Paths to scalar string fields where replacement runs. Each item must start with `.` (Vector path, e.g. `.message`, `.parsed_data.token`).

    Examples:

    ```
    labels: ".message"
    ```

    ```
    labels: ".parsed_data.line"
    ```

    - **Element of the array**
      string

      Pattern: `^\.[a-zA-Z0-9_\[\]\\\.\-"/:']+$`

  - **spec.transformations.replaceValue.source**
    string

    Required value

    Regular expression to search in each scalar field value.

    Pattern: `^[a-zA-Z0-9_\s\\\[\](){}?*+.^$\-=!:,'"|<>~#/@]+$`

Examples:

```
source: "\\d+"
```

```
source: "(?P<tok>token=[^\\s]+)"
```

- **spec.transformations.replaceValue.target**
  string

  Required value

  Replacement string; may use `{{name}}` for named capture groups.

  Examples:

```
target: REDACTED
```

```
target: "{{tok}}"
```

- **spec.type**
  string

  Type of a log storage backend.

  Allowed values: `Loki`, `Elasticsearch`, `Logstash`, `Vector`, `Kafka`, `Splunk`, `Socket`

- **spec.vector**
  object
  - **spec.vector.endpoint**
    string

    Required value

    An address of the Vector instance. API v2 must be used for communication between instances.

    Pattern: `^(.+):([0-9]{1,5})$`

  - **spec.vector.tls**
    object

    Configures the TLS options for outgoing connections.

    - **spec.vector.tls.caFile**
      string

      Base64-encoded CA certificate in PEM format.

    - **spec.vector.tls.clientCrt**
      object

      Configures the client certificate for outgoing connections.

      - **spec.vector.tls.clientCrt.crtFile**
        string

        Required value

        Base64-encoded certificate in PEM format.

        You must also set the `keyFile` parameter.

      - **spec.vector.tls.clientCrt.keyFile**
        string

        Required value

        Base64-encoded private key in PEM format (PKCS#8).

        You must also set the `crtFile` parameter.

      - **spec.vector.tls.clientCrt.keyPass**
        string

        Base64-encoded passphrase used to unlock the encrypted key file.

    - **spec.vector.tls.secretRef**
      object

      Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

      - **spec.vector.tls.secretRef.name**
        string

        Name of the Secret with TLS certificates.

    - **spec.vector.tls.verifyCertificate**
      boolean

Validate the TLS certificate of the remote host.

Default: `true`

- **spec.vector.tls.verifyHostname**
  boolean

  Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

  Default: `true`

Deprecated resource. Support for the resource might be removed in a later release.

Describes setting for a log storage, which you can use in many log sources.

`metadata.name` — is an upstream name, which you should use in custom resource ClusterLoggingConfig.

- **spec**
  object

  Required value

  - **spec.buffer**
    object

    Buffer parameters.

    - **spec.buffer.disk**
      object

      Disk buffer parameters.

      - **spec.buffer.disk.maxSize**
        integer or string

        The maximum size of the buffer on disk. Must be at least ~256MB (268435488 bytes).

        You can express size as a plain integer or as a fixed-point number using one of these quantity suffixes: `E`, `P`, `T`, `G`, `M`, `k`, `Ei`, `Pi`, `Ti`, `Gi`, `Mi`, `Ki`.

        More about resource quantity:

        - kubernetes quantity
        - memory resource units

        Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

        Examples:

        ```
        maxSize: 512Mi
        ```

        ```
        maxSize: 268435488
        ```

    - **spec.buffer.memory**
      object
      - **spec.buffer.memory.maxEvents**
        number

        The maximum number of events allowed in the buffer.

    - **spec.buffer.type**
      string

      Required value

      The type of buffer to use.

      Allowed values: `Disk`, `Memory`

    - **spec.buffer.whenFull**
      string

      Event handling behavior when a buffer is full.

      Default: `Block`

      Allowed values: `DropNewest`, `Block`

  - **spec.elasticsearch**
    object
    - **spec.elasticsearch.auth**
      object
      - **spec.elasticsearch.auth.awsAccessKey**
        string

        Base64-encoded AWS `ACCESS_KEY`.

      - **spec.elasticsearch.auth.awsAssumeRole**
        string

The ARN of an IAM role to assume at startup.

- **spec.elasticsearch.auth.awsRegion**
  string

  AWS region for authentication.

- **spec.elasticsearch.auth.awsSecretKey**
  string

  Base64-encoded AWS `SECRET_KEY`.

- **spec.elasticsearch.auth.password**
  string

  Base64-encoded Basic authentication password.

- **spec.elasticsearch.auth.strategy**
  string

  The authentication strategy to use.

  Default: `Basic`

  Allowed values: `Basic`, `AWS`

- **spec.elasticsearch.auth.user**
  string

  The Basic authentication user name.

- **spec.elasticsearch.dataStreamEnabled**
  boolean

  Use for storage indexes or datastreams (https://www.elastic.co/guide/en/elasticsearch/reference/master/data-streams.html).

  Datastream usage is better for logs and metrics storage but they works only for Elasticsearch >= 7.16.X.

  Default: `false`

- **spec.elasticsearch.docType**
  string

  The `doc_type` for your index data. This is only relevant for Elasticsearch <= 6.X.

  - For Elasticsearch >= 7.X you do not need this option since this version has removed `doc_type` mapping;
  - For Elasticsearch >= 6.X the recommended value is `_doc`, because using it will make it easy to upgrade to 7.X;
  - For Elasticsearch < 6.X you can't use a value starting with `_` or empty string. Use, for example, values like `logs`.

- **spec.elasticsearch.endpoint**
  string

  Required value

  Base URL of the Elasticsearch instance.

- **spec.elasticsearch.index**
  string

  Index name to write events to.

- **spec.elasticsearch.pipeline**
  string

  Name of the pipeline to apply.

- **spec.elasticsearch.tls**
  object

  Configures the TLS options for outgoing connections.

  - **spec.elasticsearch.tls.caFile**
    string

    Base64-encoded CA certificate in PEM format.

  - **spec.elasticsearch.tls.clientCrt**
    object

    Configures the client certificate for outgoing connections.

    - **spec.elasticsearch.tls.clientCrt.crtFile**
      string

      Required value

      Base64-encoded certificate in PEM format.

      You must also set the `keyFile` parameter.

    - **spec.elasticsearch.tls.clientCrt.keyFile**

string

Required value

Base64-encoded private key in PEM format (PKCS#8).

You must also set the `crtFile` parameter.

- **spec.elasticsearch.tls.clientCrt.keyPass**
  string

  Base64-encoded pass phrase used to unlock the encrypted key file.

- **spec.elasticsearch.tls.secretRef**
  object

  Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

  - **spec.elasticsearch.tls.secretRef.name**
    string

    Name of the Secret with TLS certificates.

- **spec.elasticsearch.tls.verifyCertificate**
  boolean

  Validate the TLS certificate of the remote host. Specifically the issuer is checked but not CRLs (Certificate Revocation Lists).

  Default: `true`

- **spec.elasticsearch.tls.verifyHostname**
  boolean

  Verifies that the name of the remote host matches the name specified in the remote host’s TLS certificate.

  Default: `true`

- **spec.extraLabels**
  object

  A set of labels that will be attached to each batch of events.

  You can use simple templating here: `{{ app }}`.

  There are some reserved keys:

  - parsed_data
  - pod
  - pod_labels_*
  - pod_ip
  - namespace
  - image
  - container
  - node
  - pod_owner

  More about field path notation…

  Example:

  ```
  extraLabels:
    forwarder: vector
    key: value
    app_info: "{{ app }}"
    array_member: "{{ array[0] }}"
    symbol_escating_value: "{{ pay\\.day }}"
  ```

- **spec.kafka**
  object
  - **spec.kafka.bootstrapServers**
    array of strings

    Required value

    A list of host and port pairs that are the addresses of the Kafka brokers in a “bootstrap” Kafka cluster that a Kafka client connects to initially to bootstrap itself.

    Default: `[]`

    Example:

    ```
    bootstrapServers:
    - 10.14.22.123:9092
    - 10.14.23.332:9092
    ```

- **Element of the array**
  string

  Pattern: `^(.+)\:\d{1,5}$`

- **spec.kafka.encoding**
  object

  How to encode the message.

  - **spec.kafka.encoding.cef**
    object

    CEF-specific configuration fields. Only applicable when codec is set to `CEF`.

    - **spec.kafka.encoding.cef.deviceProduct**
      string

      Default: `log-shipper-agent`

    - **spec.kafka.encoding.cef.deviceVendor**
      string

      Default: `Deckhouse`

    - **spec.kafka.encoding.cef.deviceVersion**
      string

      Default: `1`

  - **spec.kafka.encoding.codec**
    string

    Available encoding formats.

    Default: `JSON`

    Allowed values: `JSON`, `CEF`

- **spec.kafka.keyField**
  string

  Allows to set the key_field.

  Examples:

  ```
  keyField: host
  ```

  ```
  keyField: node
  ```

  ```
  keyField: namespace
  ```

  ```
  keyField: parsed_data.app_info
  ```

- **spec.kafka.sasl**
  object

  Configuration for SASL authentication when interacting with Kafka.

  - **spec.kafka.sasl.mechanism**
    string

    Required value

    The SASL mechanism to use. Only PLAIN and SCRAM-based mechanisms are supported.

    Allowed values: `PLAIN`, `SCRAM-SHA-256`, `SCRAM-SHA-512`

  - **spec.kafka.sasl.password**
    string

    Required value

    The SASL password.

    Example:

    ```
    password: qwerty
    ```

  - **spec.kafka.sasl.username**
    string

    Required value

    The SASL username.

    Example:

```
username: username
```

- **spec.kafka.tls**
  object

  Configures the TLS options for outgoing connections.

  - **spec.kafka.tls.caFile**
    string

    Base64-encoded CA certificate in PEM format.

  - **spec.kafka.tls.clientCrt**
    object

    Configures the client certificate for outgoing connections.

    - **spec.kafka.tls.clientCrt.crtFile**
      string

      Required value

      Base64-encoded certificate in PEM format.

      You must also set the `keyFile` parameter.

    - **spec.kafka.tls.clientCrt.keyFile**
      string

      Required value

      Base64-encoded private key in PEM format (PKCS#8).

      You must also set the `crtFile` parameter.

    - **spec.kafka.tls.clientCrt.keyPass**
      string

      Base64-encoded pass phrase used to unlock the encrypted key file.

  - **spec.kafka.tls.secretRef**
    object

    Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

    - **spec.kafka.tls.secretRef.name**
      string

      Name of the Secret with TLS certificates.

  - **spec.kafka.tls.verifyCertificate**
    boolean

    Validate the TLS certificate of the remote host.

    Default: `true`

  - **spec.kafka.tls.verifyHostname**
    boolean

    Verifies that the name of the remote host matches the name specified in the remote host’s TLS certificate.

    Default: `true`

- **spec.kafka.topic**
  string

  Required value

  The Kafka topic name to write events to. This parameter supports template syntax, which enables you to use dynamic per-event values.

  Examples:

  ```
  topic: logs
  ```

  ```
  topic: logs-{{unit}}-%Y-%m-%d
  ```

- **spec.logstash**
  object
  - **spec.logstash.endpoint**
    string

    Required value

    Base URL of the Logstash instance.

- **spec.logstash.tls**
  object

  Configures the TLS options for outgoing connections.

  - **spec.logstash.tls.caFile**
    string

    Base64-encoded CA certificate in PEM format.

  - **spec.logstash.tls.clientCrt**
    object

    Configures the client certificate for outgoing connections.

    - **spec.logstash.tls.clientCrt.crtFile**
      string

      Required value

      Base64-encoded certificate in PEM format.

      You must also set the `keyFile` parameter.

    - **spec.logstash.tls.clientCrt.keyFile**
      string

      Required value

      Base64-encoded private key in PEM format (PKCS#8).

      You must also set the `crtFile` parameter.

    - **spec.logstash.tls.clientCrt.keyPass**
      string

      Base64-encoded pass phrase used to unlock the encrypted key file.

  - **spec.logstash.tls.secretRef**
    object

    Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

    - **spec.logstash.tls.secretRef.name**
      string

      Name of the Secret with TLS certificates.

  - **spec.logstash.tls.verifyCertificate**
    boolean

    Validate the TLS certificate of the remote host.

    Default: `true`

  - **spec.logstash.tls.verifyHostname**
    boolean

    Verifies that the name of the remote host matches the name specified in the remote host’s TLS certificate.

    Default: `true`

- **spec.loki**
  object
  - **spec.loki.auth**
    object
    - **spec.loki.auth.password**
      string

      Base64-encoded Basic authentication password.

    - **spec.loki.auth.strategy**
      string

      The authentication strategy to use.

      Default: `Basic`

      Allowed values: `Basic`, `Bearer`

    - **spec.loki.auth.token**
      string

      The token to use for Bearer authentication.

    - **spec.loki.auth.user**
      string

      The Basic authentication user name.

- **spec.loki.endpoint**
  string

  Required value

  Base URL of the Loki instance.

  > **i** Agent automatically adds `/loki/api/v1/push` into URL during data transmission.

- **spec.loki.tenantID**
  string

  ID of a tenant.

  This option is used only for GrafanaCloud. When running Loki locally, a tenant ID is not required.

- **spec.loki.tls**
  object

  Configures the TLS options for outgoing connections.

  - **spec.loki.tls.caFile**
    string

    Base64-encoded CA certificate in PEM format.

  - **spec.loki.tls.clientCrt**
    object

    Configures the client certificate for outgoing connections.

    - **spec.loki.tls.clientCrt.crtFile**
      string

      Required value

      Base64-encoded certificate in PEM format.

      You must also set the `keyFile` parameter.

    - **spec.loki.tls.clientCrt.keyFile**
      string

      Required value

      Base64-encoded private key in PEM format (PKCS#8).

      You must also set the `crtFile` parameter.

    - **spec.loki.tls.clientCrt.keyPass**
      string

      Base64-encoded pass phrase used to unlock the encrypted key file.

  - **spec.loki.tls.secretRef**
    object

    Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

    - **spec.loki.tls.secretRef.name**
      string

      Name of the Secret with TLS certificates.

  - **spec.loki.tls.verifyCertificate**
    boolean

    Validate the TLS certificate of the remote host.

    If set to `false`, the certificate is not checked in the Certificate Revocation Lists.

    Default: `true`

  - **spec.loki.tls.verifyHostname**
    boolean

    Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

    Default: `true`

- **spec.rateLimit**
  object

  Parameter for limiting the flow of events.

  - **spec.rateLimit.excludes**
    array of objects

List of excludes for keyField.

Only NOT matched log entries would be rate limited.

Examples:

```
excludes:
  field: tier
  operator: Exists
```

```
excludes:
  field: foo
  operator: NotIn
  values:
  - dev
  - 42
  - 'true'
  - '3.14'
```

```
excludes:
  field: bar
  operator: Regex
  values:
  - "^abc"
  - "^\\d.+$"
```

- **spec.rateLimit.excludes.field**
  string

  Required value

  Field name for filtering.

- **spec.rateLimit.excludes.operator**
  string

  Required value

  Operator for log field comparations:

  - `In` — finds a substring in a string.
  - `NotIn` — is a negative version of the `In` operator.
  - `Regex` — is trying to match regexp over the field; only log events with matching fields will pass.
  - `NotRegex` — is a negative version of the `Regex` operator; log events without fields or with not matched fields will pass.
  - `Exists` — drops log event if it contains some fields.
  - `DoesNotExist` — drops log event if it does not contain some fields.

  Allowed values: `In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`

- **spec.rateLimit.excludes.values**
  array

  Array of values or regexes for corresponding operations. Does not work for `Exists` and `DoesNotExist` operations.

  Fields a with float or boolean values will be converted to strings during comparison.

- **spec.rateLimit.keyField**
  string

  The name of the log field whose value will be hashed to determine if the event should be rate limited.

- **spec.rateLimit.linesPerMinute**
  number

  Required value

  The number of records per minute.

- **spec.socket**
  object
  - **spec.socket.address**
    string

    Required value

    Address of the socket.

    Pattern: `^.*:[1-9][0-9]+$`

  - **spec.socket.encoding**
    object

    How to encode the message.

    - **spec.socket.encoding.cef**

object

CEF-specific configuration fields. Only applicable when codec is set to `CEF`.

- **spec.socket.encoding.cef.deviceProduct**
  string

  Default: `log-shipper-agent`

- **spec.socket.encoding.cef.deviceVendor**
  string

  Default: `Deckhouse`

- **spec.socket.encoding.cef.deviceVersion**
  string

  Default: `1`

- **spec.socket.encoding.codec**
  string

  Available encoding formats.

  Default: `JSON`

  Allowed values: `Text`, `JSON`, `Syslog`, `CEF`, `GELF`

- **spec.socket.mode**
  string

  Required value

  Allowed values: `TCP`, `UDP`

- **spec.socket.tcp**
  object
  - **spec.socket.tcp.tls**
    object

    Configures the TLS options for outgoing connections.

    - **spec.socket.tcp.tls.caFile**
      string

      Base64-encoded CA certificate in PEM format.

    - **spec.socket.tcp.tls.clientCrt**
      object

      Configures the client certificate for outgoing connections.

      - **spec.socket.tcp.tls.clientCrt.crtFile**
        string

        Required value

        Base64-encoded certificate in PEM format.

        You must also set the `keyFile` parameter.

      - **spec.socket.tcp.tls.clientCrt.keyFile**
        string

        Required value

        Base64-encoded private key in PEM format (PKCS#8).

        You must also set the `crtFile` parameter.

      - **spec.socket.tcp.tls.clientCrt.keyPass**
        string

        Base64-encoded pass phrase used to unlock the encrypted key file.

    - **spec.socket.tcp.tls.secretRef**
      object

      Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

      - **spec.socket.tcp.tls.secretRef.name**
        string

        Name of the Secret with TLS certificates.

  - **spec.socket.tcp.verifyCertificate**
    boolean

    Validate the TLS certificate of the remote host.

    If set to `false`, the certificate is not checked in the Certificate Revocation Lists.

Default: `true`

- **spec.socket.tcp.verifyHostname**
  boolean

  Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

  Default: `true`

- **spec.splunk**
  object
  - **spec.splunk.endpoint**
    string

    Required value

    Base URL of the Splunk instance.

    Example:

    ```
    endpoint: https://http-inputs-hec.splunkcloud.com
    ```

  - **spec.splunk.index**
    string

    Index name to write events to.

  - **spec.splunk.tls**
    object

    Configures the TLS options for outgoing connections.

    - **spec.splunk.tls.caFile**
      string

      Base64-encoded CA certificate in PEM format.

    - **spec.splunk.tls.clientCrt**
      object

      Configures the client certificate for outgoing connections.

      - **spec.splunk.tls.clientCrt.crtFile**
        string

        Required value

        Base64-encoded certificate in PEM format.

        You must also set the `keyFile` parameter.

      - **spec.splunk.tls.clientCrt.keyFile**
        string

        Required value

        Base64-encoded private key in PEM format (PKCS#8).

        You must also set the `crtFile` parameter.

      - **spec.splunk.tls.clientCrt.keyPass**
        string

        Base64-encoded pass phrase used to unlock the encrypted key file.

    - **spec.splunk.tls.secretRef**
      object

      Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

      - **spec.splunk.tls.secretRef.name**
        string

        Name of the Secret with TLS certificates.

    - **spec.splunk.tls.verifyCertificate**
      boolean

      Validate the TLS certificate of the remote host.

      Default: `true`

    - **spec.splunk.tls.verifyHostname**
      boolean

      Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

      Default: `true`

- **spec.splunk.token**
  string

  Required value

  Default Splunk HEC token. If an event has a token set in its metadata, it will have priority over the one set here.

- **spec.transformations**
  array of objects

  List of transformations that can be applied to logs before sending them to the destination.

  Available options:

  - `ReplaceKeys`: Replaces `source` with `target` in the specified label keys.

    Example:

    ```
    transformations:
    - action: ReplaceKeys
      replaceKeys:
        source: "."
        target: "_"
        labels:
          - .pod_labels
    ```

  - `ParseMessage`: Converts the `message` field from the format specified in `sourceFormat` into an object. If conversion fails, the `message` field remains unchanged.

    Examples:

    ```
    transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: String
        string:
          regex: "^(?P<msg>.*)$"
          setLabels:
            msg: "{{ msg }}"
    ```

    ```
    transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: Klog
    - action: ParseMessage
      parseMessage:
        sourceFormat: JSON
        json:
          depth: 1
    - action: ParseMessage
      parseMessage:
        sourceFormat: String
        string:
          regex: "^(?P<msg>.*)$"
          setLabels:
            msg: "{{ msg }}"
    ```

  - `DropLabels`: Removes the specified labels.

    Example:

    ```
    transformations:
    - action: DropLabels
      dropLabels:
        labels:
          - .first
          - .second
    ```

  - **spec.transformations.action**
    string

    Type of a transformation.

    Allowed values: `ReplaceKeys`, `ParseMessage`, `DropLabels`, `AddLabels`, `ReplaceValue`

  - **spec.transformations.addLabels**
    object
    - **spec.transformations.addLabels.setLabels**
      object

Required value

- **spec.transformations.addLabels.when**
  array of strings
  - **Element of the array**
    string

    Pattern: `^[a-zA-Z0-9_\s\\\[\](){}?*+.^$\-=!:,'"|<>~#/@]+$`

- **spec.transformations.dropLabels**
  object

  Removes the specified labels.

  - **spec.transformations.dropLabels.labels**
    array of strings

    List of labels to remove.

    - **Element of the array**
      string

      Pattern: `^\.[a-zA-Z0-9_\[\]\\\.\-]+$`

  - **spec.transformations.dropLabels.paths**
    array of objects

    List of path entries matching v1alpha2 dropLabels.labels.

    - **spec.transformations.dropLabels.paths.keepKeys**
      array of strings
      - **Element of the array**
        string

        Pattern: `^[a-zA-Z0-9_.\-/]+$`

    - **spec.transformations.dropLabels.paths.label**
      string

      Required value

      Pattern: `^\.[a-zA-Z0-9_\[\]\\\.\-]+$`

- **spec.transformations.parseMessage**
  object

  Converts the `message` field into an object.

  - **spec.transformations.parseMessage.json**
    object
    - **spec.transformations.parseMessage.json.depth**
      integer

      Depth for JSON parsing.

      Allowed values: `1 <= X <= 128`

  - **spec.transformations.parseMessage.sourceFormat**
    string

    Required value

    Message format for converting into an object.

    Allowed values: `String`, `JSON`, `Klog`, `SysLog`, `CLF`, `Logfmt`

  - **spec.transformations.parseMessage.string**
    object
    - **spec.transformations.parseMessage.string.regex**
      string

      Regular expression with named capture groups.

      Pattern: `^[a-zA-Z0-9_\s\\\[\](){}?*+.^$\-=!:,'\"|<>~#/@]+$`

    - **spec.transformations.parseMessage.string.setLabels**
      object

      Maps output keys to literals or `{{ group_name }}` templates.

    - **spec.transformations.parseMessage.string.targetField**
      string

      Target field where the structured message is placed.

      Pattern: `^[a-zA-Z0-9_\\\.\-]+$`

  - **spec.transformations.parseMessage.targetLabel**
    string

    Destination path for the parsed value.

    Default: `.message`

Pattern: `^(\.|\.[a-zA-Z0-9_\[\]\\\.\-"/:']+)$`

- **spec.transformations.replaceKeys**
  object

  Recursive replacement of all matches of the `source` pattern with the `target` value in the specified labels.

  - **spec.transformations.replaceKeys.labels**
    array of strings

    Required value

    List of labels in which the recursive replacement will be performed.

    - **Element of the array**
      string

      Pattern: `^\.[a-zA-Z0-9_\[\]\\\.\-]+$`

  - **spec.transformations.replaceKeys.source**
    string

    Required value

    Pattern used to find matches. Can be a static string or a regular expression.

  - **spec.transformations.replaceKeys.target**
    string

    Value that replaces all matches of the pattern.

    Default:

- **spec.transformations.replaceValue**
  object
  - **spec.transformations.replaceValue.labels**
    array of strings

    Required value

    - **Element of the array**
      string

      Pattern: `^\.[a-zA-Z0-9_\[\]\\\.\-"/:']+$`

  - **spec.transformations.replaceValue.source**
    string

    Required value

    Pattern: `^[a-zA-Z0-9_\s\\\[\](){}?*+.^$\-=!:,'"|<>~#/@]+$`

  - **spec.transformations.replaceValue.target**
    string

    Required value

- **spec.type**
  string

  Type of a log storage backend.

  Allowed values: `Loki`, `Elasticsearch`, `Logstash`, `Vector`, `Kafka`, `Splunk`, `Socket`

- **spec.vector**
  object
  - **spec.vector.endpoint**
    string

    Required value

    An address of the Vector instance. API v2 must be used for communication between instances.

    Pattern: `^(.+):([0-9]{1,5})$`

  - **spec.vector.tls**
    object

    Configures the TLS options for outgoing connections.

    - **spec.vector.tls.caFile**
      string

      Base64-encoded CA certificate in PEM format.

    - **spec.vector.tls.clientCrt**
      object

      Configures the client certificate for outgoing connections.

      - **spec.vector.tls.clientCrt.crtFile**
        string

        Required value

Base64-encoded certificate in PEM format.

You must also set the `keyFile` parameter.

- **spec.vector.tls.clientCrt.keyFile**
  string

  Required value

  Base64-encoded private key in PEM format (PKCS#8).

  You must also set the `crtFile` parameter.

- **spec.vector.tls.clientCrt.keyPass**
  string

  Base64-encoded passphrase used to unlock the encrypted key file.

- **spec.vector.tls.secretRef**
  object

  Reference to a Kubernetes Secret containing the CA certificate (ca.pem), client certificate (crt.pem), private key (key.pem) and key pass (keyPass) in Base64-encoded PEM format. If specified, TLS settings are overridden with values from the secret. Secret should be located in d8-log-shipper namespace and have `log-shipper.deckhouse.io/watch-secret: true` label.

  - **spec.vector.tls.secretRef.name**
    string

    Name of the Secret with TLS certificates.

- **spec.vector.tls.verifyCertificate**
  boolean

  Validate the TLS certificate of the remote host.

  Default: `true`

- **spec.vector.tls.verifyHostname**
  boolean

  Verifies that the name of the remote host matches the name specified in the remote host's TLS certificate.

  Default: `true`

ClusterLoggingConfig

Scope: Cluster

- v1alpha2
- v1alpha1

Describes a log source in log-pipeline.

Each custom resource `ClusterLoggingConfig` describes rules for log fetching from cluster.

- **spec**
  object

  Required value

  - **spec.destinationRefs**
    array of strings

    Required value

    Array of `ClusterLogDestination` custom resource names which this source will output with.

    Fields with float or boolean values will be converted to strings.

  - **spec.file**
    object

    Describes a rule for collecting logs from files on a node.

    - **spec.file.exclude**
      array of strings

      Array of file patterns to exclude when collecting logs.

      Wildcards are supported.

      Examples:

      ```
      exclude: "/var/log/nginx/error.log"
      ```

      ```
      exclude: "/var/log/audit.log"
      ```

    - **spec.file.include**
      array of strings

Array of file patterns to include.

Wildcards are supported

Examples:

```
include: "/var/log/*.log"
```

```
include: "/var/log/nginx/*.log"
```

- **spec.file.lineDelimiter**
  string

  String sequence used to separate one file line from another.

  Example:

  ```
  lineDelimiter: "\\r\\n"
  ```

- **spec.kubernetesPods**
  object

  Describes a rule for collecting logs from the cluster’s pods.

  - **spec.kubernetesPods.keepDeletedFilesOpenedFor**
    string

    Specifies the time to keep deleted files opened for reading. Vector will keep pods metadata for this time as well to read logs from deleted pods. This option is useful in cases of a log storage unavailability or a network partition. Vector will keep log files opened until finally sending them to the destination.

    Enabling this option may affect the resource consumption of the Vector and also flood a disk with deleted logs. Use it with caution.

    The format is a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

    Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

  - **spec.kubernetesPods.labelSelector**
    object

    Specifies the label selector to filter Pods with.

    You can get more into here.

    - **spec.kubernetesPods.labelSelector.matchExpressions**
      array of objects

      List of label expressions for Pods.

      Example:

      ```
      matchExpressions:
      - key: tier
        operator: In
        values:
        - production
        - staging
      - key: tier
        operator: NotIn
        values:
        - production
      ```

      - **spec.kubernetesPods.labelSelector.matchExpressions.key**
        string

        A label name.

      - **spec.kubernetesPods.labelSelector.matchExpressions.operator**
        string

        A comparison operator.

        Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

      - **spec.kubernetesPods.labelSelector.matchExpressions.values**
        array of strings

        A label value.

        - **Element of the array**
          string

          Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

          Length: `1..63`

    - **spec.kubernetesPods.labelSelector.matchLabels**

object

List of labels which Pod should have.

Example:

```
matchLabels:
  foo: bar
  baz: who
```

- **spec.kubernetesPods.namespaceSelector**
  object

  Specifies the namespace selector to filter Pods with.

  The filter uses the `labelSelector` parameter to determine the namespaces from which logs should be collected.

  - **spec.kubernetesPods.namespaceSelector.labelSelector**
    object

    Specifies the label selector to filter namespaces from which logs should be collected.

    You can get more into here.

    - **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions**
      array of objects

      List of label expressions that a namespace should have to qualify for the filter condition.

      Example:

      ```
      matchExpressions:
      - key: tier
        operator: In
        values:
        - production
        - staging
      ```

      - **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions.key**
        string

        Required value

        A label name.

      - **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions.operator**
        string

        Required value

        A comparison operator.

        Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

      - **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions.values**
        array of strings

        A label value.

        - **Element of the array**
          string

          Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

          Length: `1..63`

    - **spec.kubernetesPods.namespaceSelector.labelSelector.matchLabels**
      object

      List of labels that a namespace should have to qualify for the filter condition.

      Example:

      ```
      matchLabels:
        foo: bar
        baz: who
      ```

- **spec.labelFilter**
  array of objects

  Rules to filter log lines by their metadata labels.

  Example:

```
labelFilter:
- field: container
  operator: In
  values:
  - nginx
- field: pod_labels.tier
  operator: Regex
  values:
  - prod-.+
  - stage-.+
```

- **spec.labelFilter.field**
  string

  Required value

  Label name for filtering.

  Must not be empty.

  Pattern: `.+`

- **spec.labelFilter.operator**
  string

  Required value

  Operator for log field comparations:

  - `In` — finds a substring in a string.
  - `NotIn` — is a negative version of the `In` operator.
  - `Regex` — is trying to match regexp over the field; only log events with matching fields will pass.
  - `NotRegex` — is a negative version of the `Regex` operator; log events without fields or with not matched fields will pass.
  - `Exists` — drops log event if it contains some fields.
  - `DoesNotExist` — drops log event if it does not contain some fields.

  Allowed values: `In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`

- **spec.labelFilter.values**
  array

  Array of values or regexes for corresponding operations. Does not work for `Exists` and `DoesNotExist` operations.

  Fields a with float or boolean values will be converted to strings during comparison.

- **spec.logFilter**
  array of objects

  A list of filters for logs that are applied to messages in JSON format.

  Only matched lines would be stored to log destination.

  Example:

```
logFilter:
- field: tier
  operator: Exists
- field: foo
  operator: NotIn
  values:
  - dev
  - 42
  - 'true'
  - '3.14'
- field: bar
  operator: Regex
  values:
  - "^abc"
  - "^\\d.+$"
```

  - **spec.logFilter.field**
    string

    Required value

    Field name for filtering. It should be empty for non-JSON messages.

  - **spec.logFilter.operator**
    string

    Required value

    Operator for log field comparations:

- `In` — finds a substring in a string.
- `NotIn` — is a negative version of the `In` operator.
- `Regex` — is trying to match regexp over the field; only log events with matching fields will pass.
- `NotRegex` — is a negative version of the `Regex` operator; log events without fields or with not matched fields will pass.
- `Exists` — drops log event if it contains some fields.
- `DoesNotExist` — drops log event if it does not contain some fields.

Allowed values: `In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`

- **spec.logFilter.values**
  array

  Array of values or regexes for corresponding operations. Does not work for `Exists` and `DoesNotExist` operations.

  Fields a with float or boolean values will be converted to strings during comparison.

- **spec.multilineParser**
  object

  Multiline parser for different patterns.

  - **spec.multilineParser.custom**
    object

    Multiline parser custom regex rules.

    - **spec.multilineParser.custom.endsWhen**
      object

      It’s a condition to distinguish the last log line of multiline log.

      - **spec.multilineParser.custom.endsWhen.notRegex**
        string

        Regex string, which treats as match only strings that DOESN’T match regex.

      - **spec.multilineParser.custom.endsWhen.regex**
        string

        Regex string, which treats as match only strings that match regex.

    - **spec.multilineParser.custom.startsWhen**
      object

      It’s a condition to distinguish the first log line of multiline log.

      - **spec.multilineParser.custom.startsWhen.notRegex**
        string

        Regex string, which treats as match only strings that DOESN’T match regex.

      - **spec.multilineParser.custom.startsWhen.regex**
        string

        Regex string, which treats as match only strings that match regex.

  - **spec.multilineParser.type**
    string

    Required value

    Parser types:

    - `None` — do not parse logs.
    - `General` — tries to match general multiline logs with space or tabulation on extra lines.
    - `Backslash` — tries to match bash style logs with backslash on all lines except the last event line.
    - `LogWithTime` — tries to detect events by timestamp.
    - `MultilineJSON` — tries to match JSON logs, assuming the event starts with the `{` symbol.
    - `Custom` - tries to match logs with the user provided regex in `spec.multilineParser.custom` field.

    Default: `None`

    Allowed values: `None`, `General`, `Backslash`, `LogWithTime`, `MultilineJSON`, `Custom`

- **spec.type**
  string

  Required value

  Set on of possible input sources.

  `KubernetesPods` source reads logs from Kubernetes Pods.

  `File` source reads local file from node filesystem.

  Allowed values: `KubernetesPods`, `File`

Deprecated resource. Support for the resource might be removed in a later release.

Describes a log source in log-pipeline.

Each custom resource `ClusterLoggingConfig` describes rules for log fetching from cluster.

- **spec**
  object

  Required value

  - **spec.destinationRefs**
    array of strings

    Required value

    Array of `ClusterLogDestination` custom resource names which this source will output with.

    Fields with float or boolean values will be converted to strings.

  - **spec.file**
    object

    Describes a rule for collecting logs from files on a node.

    - **spec.file.exclude**
      array of strings

      Array of file patterns to exclude when collecting logs.

      Wildcards are supported.

      Examples:

      ```
      exclude: "/var/log/nginx/error.log"
      ```

      ```
      exclude: "/var/log/audit.log"
      ```

    - **spec.file.include**
      array of strings

      Array of file patterns to include.

      Wildcards are supported

      Examples:

      ```
      include: "/var/log/*.log"
      ```

      ```
      include: "/var/log/nginx/*.log"
      ```

    - **spec.file.lineDelimiter**
      string

      String sequence used to separate one file line from another.

      Example:

      ```
      lineDelimiter: "\\r\\n"
      ```

  - **spec.kubernetesPods**
    object

    Describes a rule for collecting logs from the cluster’s pods.

    - **spec.kubernetesPods.keepDeletedFilesOpenedFor**
      string

      Specifies the time to keep deleted files opened for reading. Vector will keep pods metadata for this time as well to read logs from deleted pods. This option is useful in cases of a log storage unavailability or a network partition. Vector will keep log files opened until finally sending them to the destination.

      Enabling this option may affect the resource consumption of the Vector and also flood a disk with deleted logs. Use it with caution.

      The format is a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

      Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

    - **spec.kubernetesPods.labelSelector**
      object

      Specifies the label selector to filter Pods with.

      You can get more into here.

      - **spec.kubernetesPods.labelSelector.matchExpressions**
        array of objects

        List of label expressions for Pods.

        Example:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
- key: tier
  operator: NotIn
  values:
  - production
```

- **spec.kubernetesPods.labelSelector.matchExpressions.key**
  string

  A label name.

- **spec.kubernetesPods.labelSelector.matchExpressions.operator**
  string

  A comparison operator.

  Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.kubernetesPods.labelSelector.matchExpressions.values**
  array of strings

  A label value.

  - **Element of the array**
    string

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

    Length: `1..63`

- **spec.kubernetesPods.labelSelector.matchLabels**
  object

  List of labels which Pod should have.

  Example:

```
matchLabels:
  foo: bar
  baz: who
```

- **spec.kubernetesPods.namespaceSelector**
  object

  Specifies the namespace selector to filter Pods with.

  The filter can use one of the three available ways to set the condition (parameters `matchNames`, `excludeNames`, `labelSelector`)

  - **spec.kubernetesPods.namespaceSelector.excludeNames**
    array of strings

    A list of namespaces, from the pods of which you need to exclude the collection of logs, but collect from the rest.

  - **spec.kubernetesPods.namespaceSelector.labelSelector**
    object

    Specifies the label selector to filter namespaces from which logs should be collected.

    You can get more into here.

    - **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions**
      array of objects

      List of label expressions that a namespace should have to qualify for the filter condition.

      Example:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
```

      - **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions.key**
        string

        A label name.

      - **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions.operator**

string

A comparison operator.

Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.kubernetesPods.namespaceSelector.labelSelector.matchExpressions.values**
  array of strings

  A label value.

  - **Element of the array**
    string

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

    Length: `1..63`

- **spec.kubernetesPods.namespaceSelector.labelSelector.matchLabels**
  object

  List of labels that a namespace should have to qualify for the filter condition.

  Example:

  ```
  matchLabels:
    foo: bar
    baz: who
  ```

- **spec.kubernetesPods.namespaceSelector.matchNames**
  array of strings

  A list of namespaces from whose pods logs should be collected.

- **spec.labelFilter**
  array of objects

  Rules to filter log lines by their metadata labels.

  Example:

  ```
  labelFilter:
  - field: container
    operator: In
    values:
    - nginx
  - field: pod_labels.tier
    operator: Regex
    values:
    - prod-.+
    - stage-.+
  ```

  - **spec.labelFilter.field**
    string

    Required value

    Label name for filtering.

    Must not be empty.

    Pattern: `.+`

  - **spec.labelFilter.operator**
    string

    Required value

    Operator for log field comparations:

    - `In` — finds a substring in a string.
    - `NotIn` — is a negative version of the `In` operator.
    - `Regex` — is trying to match regexp over the field; only log events with matching fields will pass.
    - `NotRegex` — is a negative version of the `Regex` operator; log events without fields or with not matched fields will pass.
    - `Exists` — drops log event if it contains some fields.
    - `DoesNotExist` — drops log event if it does not contain some fields.

    Allowed values: `In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`

  - **spec.labelFilter.values**
    array

    Array of values or regexes for corresponding operations. Does not work for `Exists` and `DoesNotExist` operations.

    Fields a with float or boolean values will be converted to strings during comparison.

- **spec.logFilter**

array of objects

A list of filters for logs that are applied to messages in JSON format.

Only matched lines would be stored to log destination.

Example:

```
logFilter:
- field: tier
  operator: Exists
- field: foo
  operator: NotIn
  values:
  - dev
  - 42
  - 'true'
  - '3.14'
- field: bar
  operator: Regex
  values:
  - "^abc"
  - "^\\d.+$"
```

- **spec.logFilter.field**
  string

  Required value

  Field name for filtering. It should be empty for non-JSON messages.

- **spec.logFilter.operator**
  string

  Required value

  Operator for log field comparations:

  - `In` — finds a substring in a string.
  - `NotIn` — is a negative version of the `In` operator.
  - `Regex` — is trying to match regexp over the field; only log events with matching fields will pass.
  - `NotRegex` — is a negative version of the `Regex` operator; log events without fields or with not matched fields will pass.
  - `Exists` — drops log event if it contains some fields.
  - `DoesNotExist` — drops log event if it does not contain some fields.

  Allowed values: `In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`

- **spec.logFilter.values**
  array

  Array of values or regexes for corresponding operations. Does not work for `Exists` and `DoesNotExist` operations.

  Fields a with float or boolean values will be converted to strings during comparison.

- **spec.multilineParser**
  object

  Multiline parser for different patterns.

  - **spec.multilineParser.custom**
    object

    Multiline parser custom regex rules.

    - **spec.multilineParser.custom.endsWhen**
      object

      It’s a condition to distinguish the last log line of multiline log.

      - **spec.multilineParser.custom.endsWhen.notRegex**
        string

        Regex string, which treats as match only strings that DOESN’T match regex.

      - **spec.multilineParser.custom.endsWhen.regex**
        string

        Regex string, which treats as match only strings that match regex.

    - **spec.multilineParser.custom.startsWhen**
      object

      It’s a condition to distinguish the first log line of multiline log.

      - **spec.multilineParser.custom.startsWhen.notRegex**
        string

Regex string, which treats as match only strings that DOESN’T match regex.

- **spec.multilineParser.custom.startsWhen.regex**
  string

  Regex string, which treats as match only strings that match regex.

- **spec.multilineParser.type**
  string

  Required value

  Parser types:

  - `None` — do not parse logs.
  - `General` — tries to match general multiline logs with space or tabulation on extra lines.
  - `Backslash` — tries to match bash style logs with backslash on all lines except the last event line.
  - `LogWithTime` — tries to detect events by timestamp.
  - `MultilineJSON` — tries to match JSON logs, assuming the event starts with the `{` symbol.
  - `Custom` - tries to match logs with the user provided regex in `spec.multilineParser.custom` field.

  Default: `None`

  Allowed values: `None`, `General`, `Backslash`, `LogWithTime`, `MultilineJSON`, `Custom`

- **spec.type**
  string

  Required value

  Set on of possible input sources.

  `KubernetesPods` source reads logs from Kubernetes Pods.

  `File` source reads local file from node filesystem.

  Allowed values: `KubernetesPods`, `File`

PodLoggingConfig

Scope: Namespaced
Version: v1alpha1

Custom resource for namespaced Kubernetes source.

Each custom resource `PodLoggingConfig` describes rules for log fetching from specified namespace.

- **spec**
  object

  Required value

  - **spec.clusterDestinationRefs**
    array of strings

    Required value

    Array of `ClusterLogDestination` custom resource names which this source will output with.

  - **spec.keepDeletedFilesOpenedFor**
    string

    Specifies the time to keep deleted files opened for reading. Vector will keep pods metadata for this time as well to read logs from deleted pods. This option is useful in cases of a log storage unavailability or a network partition. Vector will keep log files opened until finally sending them to the destination.

    Enabling this option may affect the resource consumption of the Vector and also flood a disk with deleted logs. Use it with caution.

    The format is a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

    Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

  - **spec.labelFilter**
    array of objects

    Rules to filter log lines by their metadata labels.

    Example:

```
labelFilter:
- field: container
  operator: In
  values:
  - nginx
- field: pod_labels.tier
  operator: Regex
  values:
  - prod-.+
  - stage-.+
- field: message
  operator: Regex
  values:
  - ".*search_text.*"
```

- **spec.labelFilter.field**
  string

  Required value

  Label name for filtering.

  Must not be empty.

  Pattern: `.+`

- **spec.labelFilter.operator**
  string

  Required value

  Operator for log field comparations:

  - `In` — finds a substring in a string.
  - `NotIn` — is a negative version of the `In` operator.
  - `Regex` — is trying to match regexp over the field; only log events with matching fields will pass.
  - `NotRegex` — is a negative version of the `Regex` operator; log events without fields or with not matched fields will pass.
  - `Exists` — drops log event if it contains some fields.
  - `DoesNotExist` — drops log event if it does not contain some fields.

  Allowed values: `In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`

- **spec.labelFilter.values**
  array

  Array of values or regexes for corresponding operations. Does not work for `Exists` and `DoesNotExist` operations.

  Fields with a float or boolean values will be converted to strings during comparison.

- **spec.labelSelector**
  object

  Specifies the label selector to filter Pods.

  For more information, see the Kubernetes documentation.

  - **spec.labelSelector.matchExpressions**
    array of objects

    List of label expressions for Pods.

    Example:

    ```
    matchExpressions:
    - key: tier
      operator: In
      values:
      - production
      - staging
    ```

    - **spec.labelSelector.matchExpressions.key**
      string

      A label name.

    - **spec.labelSelector.matchExpressions.operator**
      string

      A comparison operator.

      Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

    - **spec.labelSelector.matchExpressions.values**
      array of strings

A label value.

- **Element of the array**
  string

  Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

  Length: `1..63`

- **spec.labelSelector.matchLabels**
  object

  List of labels which Pod should have.

  Example:

  ```
  matchLabels:
    foo: bar
    baz: who
  ```

- **spec.logFilter**
  array of objects

  A list of filters for logs that are applied to messages in JSON format.

  Only matched lines would be stored to log destination.

  Example:

  ```
  logFilter:
  - field: tier
    operator: Exists
  - field: foo
    operator: NotIn
    values:
    - dev
    - 42
    - 'true'
    - '3.14'
  - field: bar
    operator: Regex
    values:
    - "^abc"
    - "^\\d.+$"
  ```

  - **spec.logFilter.field**
    string

    Required value

    Field name for filtering. It should be empty for non-JSON messages.

  - **spec.logFilter.operator**
    string

    Required value

    Operator for log field comparations:

    - `In` — finds a substring in a string.
    - `NotIn` — is a negative version of the `In` operator.
    - `Regex` — is trying to match regexp over the field; only log events with matching fields will pass.
    - `NotRegex` — is a negative version of the `Regex` operator; log events without fields or with not matched fields will pass.
    - `Exists` — drops log event if it contains some fields.
    - `DoesNotExist` — drops log event if it does not contain some fields.

    Allowed values: `In`, `NotIn`, `Regex`, `NotRegex`, `Exists`, `DoesNotExist`

  - **spec.logFilter.values**
    array

    Array of values or regexes for corresponding operations. Does not work for `Exists` and `DoesNotExist` operations.

    Fields a with float or boolean values will be converted to strings during comparison.

- **spec.multilineParser**
  object

  Multiline parser for different patterns.

  - **spec.multilineParser.custom**
    object

    Multiline parser custom regex rules.

    - **spec.multilineParser.custom.endsWhen**

object

It’s a condition to distinguish the last log line of the multiline log.

- **spec.multilineParser.custom.endsWhen.notRegex**
  string

  Regex string, which treats as match only strings that DON’T match the regex.

- **spec.multilineParser.custom.endsWhen.regex**
  string

  Regex string, which treats as match only strings that match the regex.

- **spec.multilineParser.custom.startsWhen**
  object

  It’s a condition to distinguish the first log line of multiline log.

  - **spec.multilineParser.custom.startsWhen.notRegex**
    string

    Regex string, which treats as match only strings that DON’T match the regex.

  - **spec.multilineParser.custom.startsWhen.regex**
    string

    Regex string, which treats as match only strings that match the regex.

- **spec.multilineParser.type**
  string

  Required value

  Parser types:

  - `None` — do not parse logs.
  - `General` — tries to match general multiline logs with space or tabulation on extra lines.
  - `Backslash` — tries to match bash style logs with backslash on all lines except the last event line.
  - `LogWithTime` — tries to detect events by timestamp.
  - `MultilineJSON` — tries to match JSON logs, assuming the event starts with the `{` symbol.
  - `Custom` - tries to match logs with the user provided regex in `spec.multilineParser.custom` field.

  Default: `None`

  Allowed values: `None`, `General`, `Backslash`, `LogWithTime`, `MultilineJSON`, `Custom`

FAQ

> **i** The module lifecycle stage: General Availability

How to add authorization to the *ClusterLogDestination* resource?

To add authorization parameters to the ClusterLogDestination resource, you need to:

- change the connection protocol to Loki to HTTPS
- add the auth section, in which:
  - the strategy parameter should be set to `Bearer`;
  - the token parameter should contain the `log-shipper-token` token from the `d8-log-shipper` namespace.

For example:

- ClusterLogDestination resource without authorization:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki
spec:
  type: Loki
  loki:
    endpoint: "http://loki.d8-monitoring:3100"
```

- Get the `log-shipper-token` token from the `d8-log-shipper` namespace:

```
d8 k -n d8-log-shipper get secret log-shipper-token -o jsonpath='{.data.token}' | base64 -d
```

- ClusterLogDestination resource with authorization:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki
spec:
  type: Loki
  loki:
    endpoint: "https://loki.d8-monitoring:3100"
    auth:
      strategy: "Bearer"
      token: <log-shipper-token>
    tls:
      verifyHostname: false
      verifyCertificate: false
```

Advanced usage

> **i** The module lifecycle stage: General Availability

How to enable debugging logs?

Enable the debug parameter of the module.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: log-shipper
spec:
  version: 1
  enabled: true
  settings:
    debug: true
```

Then in logs, you will find a lot of helpful information about HTTP requests, connects reusing, detailed traces, and so on.

How to get aware of logs pipelines?

To begin with, retrieve the list of pods on the desired node.

```
d8 k -n d8-log-shipper get pods -o wide | grep $node
```

Select the appropriate pod and execute commands directly from the container.

```
d8 k -n d8-log-shipper exec $pod -c vector -- COMMAND_NAME
```

All subsequent commands are assumed to be executed within the container.

See pipelines as a graph

- Execute the `vector graph` command to get the output of logs pipelines topology in the DOT format.
- Put the output to webgraphviz os similar service to render the graph.

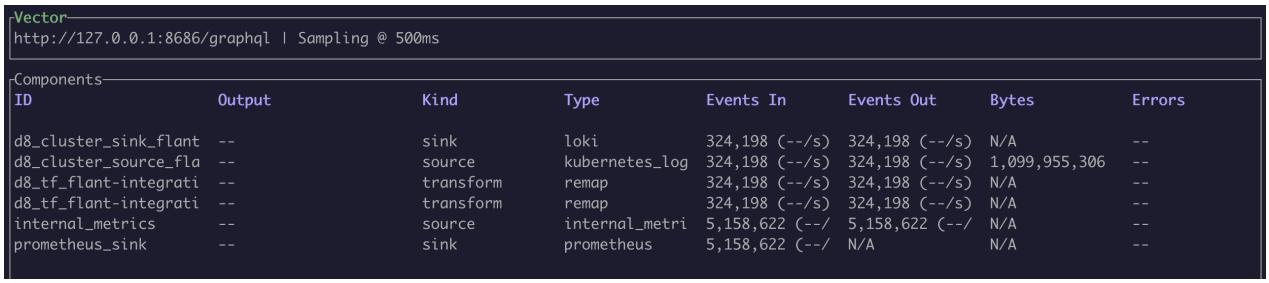
Example of the graph output for a single pipeline in ASCII format:

```
+-----------------------------------------------+
|  d8_cluster_source_flant-integration-d8-logs  |
+-----------------------------------------------+
  |
  |
  v
+-----------------------------------------------+
|      d8_tf_flant-integration-d8-logs_0        |
+-----------------------------------------------+
  |
  |
  v
+-----------------------------------------------+
|      d8_tf_flant-integration-d8-logs_1        |
+-----------------------------------------------+
  |
  |
  v
+-----------------------------------------------+
| d8_cluster_sink_flant-integration-loki-storage |
+-----------------------------------------------+
```

Investigate data processing

There is the `vector top` command to help you see how much data is going through all checkpoints of the pipeline.

Example of the output:



Get raw log samples

You may use the `vector tap` command to get input data for any channel components. The only argument to be set is the ID of the processing stage. Patterns in a glob format are also supported for selections (e.g., `d8_cluster_source_*`).

By using the following command, you can see the log's before applying the transformation rules (`d8_cluster_source_*` is the first stage of processing according to the output of the `vector graph` command):

```
vector tap 'cluster_logging_config/*'
```

Transformed logs that are received at the input of the next channel components in the chain:

```
vector tap 'transform/*'
```

You can then use the `vector vrl` interactive console to debug VRL message remap rules.

Example of a program on VRL:

```
. = {"test1": "lynx", "test2": "fox"}
del(.test2)
.
```

How to add a new source/sink support for log-shipper?

Vector in the `log-shipper` module has been built with the limited number of enabled features (to improve building speed and decrease the size of the final binary).

You can see a list of all supported features by executing the `vector list` command.

If supporting a new source/sink is required, you need to add the corresponding feature to the list of enabled features in the Dockerfile.

Examples

> **i** The module lifecycle stage: General Availability

Getting logs from all cluster Pods and sending them to Loki

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: all-logs
spec:
  type: KubernetesPods
  destinationRefs:
  - loki-storage
---
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

Specify `d8-loki` in the ClusterLoggingConfig destinationRefs field to send logs to the loki cluster module. More examples in the documentation.

Reading Pod logs from a specified namespace with a specified label and redirecting to Loki and Elasticsearch

Reading logs from `namespace=whispers` with label `app=booking` and storing them into Loki and Elasticsearch:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: whispers-booking-logs
spec:
  type: KubernetesPods
  kubernetesPods:
    namespaceSelector:
      labelSelector:
        matchExpressions:
        - key: "kubernetes.io/metadata.name"
          operator: In
          values: [whispers]
    labelSelector:
      matchLabels:
        app: booking
  destinationRefs:
  - loki-storage
  - es-storage
---
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
---
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: logs-%F
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmV0IC1uCg==
```

Creating a source in namespace and reading logs of all Pods in that NS with forwarding them to Loki

Namespaced pipeline - reading logs from `test-whispers` namespace (all Pods):

```
apiVersion: deckhouse.io/v1alpha1
kind: PodLoggingConfig
metadata:
  name: whispers-logs
  namespace: tests-whispers
spec:
  clusterDestinationRefs:
    - loki-storage
---
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

Reading only Pods in the specified namespace and having a certain label

Read logs from Pods with label `app=booking` in namespace `test-whispers`:

```
apiVersion: deckhouse.io/v1alpha1
kind: PodLoggingConfig
metadata:
  name: whispers-logs
  namespace: tests-whispers
spec:
  labelSelector:
    matchLabels:
      app: booking
  clusterDestinationRefs:
    - loki-storage
---
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

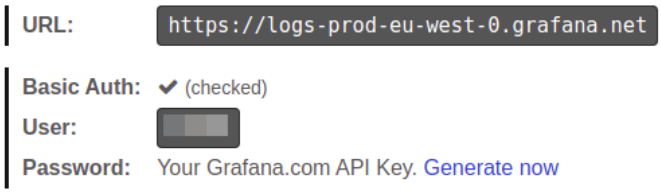
Migration from Promtail to Log-Shipper

Path `/loki/api/v1/push` has to be removed from the previously used Loki URL.

Vector will add this PATH automatically during working with Loki destination.

Working with Grafana Cloud

This documentation expects that you have created API key.



Firstly you should encode your token with base64.

```
echo -n "<YOUR-GRAFANACLOUD-TOKEN>" | base64 -w0
```

Then you can create **ClusterLogDestination**

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  loki:
    auth:
      password: PFlPVVItR1JBRkFOQUNMT1VELVRPS0VOPg==
      strategy: Basic
      user: "<YOUR-GRAFANACLOUD-USER>"
    endpoint: <YOUR-GRAFANACLOUD-URL> # For example https://logs-prod-us-central1.grafana.net or
https://logs-prod-eu-west-0.grafana.net
  type: Loki
```

Now you can create PodLogginConfig or ClusterPodLoggingConfig and send logs to **Grafana Cloud**.

Adding Loki source to Deckhouse Grafana

You can work with Loki from embedded to deckhouse Grafana. Just add **GrafanaAdditionalDatasource**

```
apiVersion: deckhouse.io/v1
kind: GrafanaAdditionalDatasource
metadata:
  name: loki
spec:
  access: Proxy
  basicAuth: false
  jsonData:
    maxLines: 5000
    timeInterval: 30s
  type: loki
  url: http://loki.loki:3100
```

Elasticsearch < 6.X usage

For Elasticsearch < 6.0 doc_type indexing should be set. Config should look like this:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    docType: "myDocType" # Set any string here. It should not start with '_'
    auth:
      strategy: Basic
      user: elastic
      password: c2VjcmV0IC1uCg==
```

Index template for Elasticsearch

It is possible to route logs to particular indexes based on metadata using index templating:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: "k8s-{{ namespace }}-%F"
```

For the above example for each Kubernetes namespace a dedicated index in Elasticsearch will be created.

This feature works well combining with `extraLabels`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://192.168.1.1:9200
    index: "k8s-{{ service }}-{{ namespace }}-%F"
  extraLabels:
    service: "{{ service_name }}"
```

1. If a log message is in JSON format, the `service_name` field of this JSON document is moved to the metadata level.
2. The new metadata field `service` is used for the index template.

Splunk integration

It is possible to send logs from Deckhouse to Splunk.

1. Endpoint must be equal to the Splunk instance name with the `8088` port and no path provided, e.g. `https://prd-p-xxxxxx.splunkcloud.com:8088`.
2. To add a token to ingest logs, go to `Setting -> Data inputs`, add a new `HTTP Event Collector` and copy a token.
3. Provide a Splunk index to store logs, e.g., `logs`.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: splunk
spec:
  type: Splunk
  splunk:
    endpoint: https://prd-p-xxxxxx.splunkcloud.com:8088
    token: xxxx-xxxx-xxxx
    index: logs
    tls:
      verifyCertificate: false
      verifyHostname: false
```

> **i** Splunk destination doesn't support pod labels for indexes. Consider exporting necessary labels with the `extraLabels` option.

```
extraLabels:
  pod_label_app: '{{ pod_labels.app }}'
```

Simple Logstash example

To send logs to Logstash, the `tcp` input should be configured on the Logstash instance side, and its codec should be set to `json`.

An example of the minimal Logstash configuration:

```
input {
  tcp {
    port => 12345
    codec => json
  }
}
output {
  stdout { codec => json }
}
```

An example of the `ClusterLogDestination` manifest:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: logstash
spec:
  type: Logstash
  logstash:
    endpoint: logstash.default:12345
```

Syslog

The following examples sets severity for the syslog messages and uses the socket destination:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: rsyslog
spec:
  type: Socket
  socket:
    mode: TCP
    address: 192.168.0.1:3000
    encoding:
      codec: Syslog
  extraLabels:
    syslog.severity: "alert"
    # the request_id field should be present in the log message
    syslog.message_id: "{{ request_id }}"
```

Graylog integration

Make sure that an incoming stream is configured in Graylog to receive messages over the TCP protocol on the specified port.

Example manifest for integration with Graylog:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: test-socket2-dest
spec:
  type: Socket
  socket:
    address: graylog.svc.cluster.local:9200
    mode: TCP
    encoding:
      codec: GELF
```

Logs in CEF Format

There is a way to format logs in CEF format using `codec: CEF`, with overriding `cef.name` and `cef.severity` based on values from the `message` field (application log) in JSON format.

In the example below, `app` and `log_level` are keys containing values for overriding:

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: siem-kafka
spec:
  extraLabels:
    cef.name: '{{ app }}'
    cef.severity: '{{ log_level }}'
  type: Kafka
  kafka:
    bootstrapServers:
      - my-cluster-kafka-brokers.kafka:9092
    encoding:
      codec: CEF
    tls:
      verifyCertificate: false
      verifyHostname: true
    topic: logs
```

You can also manually set your own values:

```
extraLabels:
  cef.name: 'TestName'
  cef.severity: '1'
```

Collect Kubernetes Events

Kubernetes Events can be collected by log-shipper if `events-exporter` is enabled in the extended-monitoring module configuration.

Enable `events-exporter` by adjusting `extended-monitoring` settings:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: extended-monitoring
spec:
  version: 1
  settings:
    events:
      exporterEnabled: true
```

Apply the following `ClusterLoggingConfig` to collect logs from the `events-exporter` Pod:

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubernetes-events
spec:
  type: KubernetesPods
  kubernetesPods:
    labelSelector:
      matchLabels:
        app: events-exporter
    namespaceSelector:
      labelSelector:
        matchExpressions:
        - key: "kubernetes.io/metadata.name"
          operator: In
          values: [d8-monitoring]
  destinationRefs:
  - loki-storage
```

Log filters

Users can filter logs by applying two filters:

- `labelFilter` — applies to the top-level metadata, e.g., container, namespace, or Pod name.
- `logFilter` — applies to fields of a message if it is in JSON format.

Collect only logs of the `nginx` container

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: nginx-logs
spec:
  type: KubernetesPods
  labelFilter:
  - field: container
    operator: In
    values: [nginx]
  destinationRefs:
  - loki-storage
```

Collect logs without strings `GET /status" 200`

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: all-logs
spec:
  type: KubernetesPods
  destinationRefs:
  - loki-storage
  labelFilter:
  - field: message
    operator: NotRegex
    values:
    - .*GET /status" 200$
```

Audit of kubelet actions

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: kubelet-audit-logs
spec:
  type: File
  file:
    include:
    - /var/log/kube-audit/audit.log
  logFilter:
  - field: userAgent
    operator: Regex
    values: ["kubelet.*"]
  destinationRefs:
  - loki-storage
```

Deckhouse system logs

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: system-logs
spec:
  type: File
  file:
    include:
    - /var/log/syslog
  labelFilter:
  - field: message
    operator: Regex
    values:
    - .*d8-kubelet-forker.*
    - .*containerd.*
    - .*bashible.*
    - .*kernel.*
  destinationRefs:
  - loki-storage
```

> **i** If you need logs from only one or from a small group of a Pods, try to use the kubernetesPods settings to reduce the number of reading filed. Do not use highly grained filters to read logs from a single pod.

Log transformations

In a ClusterLogDestination resource, you can chain transformations in order. The following actions are supported: `ParseMessage`, `ReplaceKeys`, `ReplaceValue`, `AddLabels`, and `DropLabels`.

Transforming logs into a structured object

You can use the `ParseMessage` transformation to convert a string in the `message` field into a structured object. For `sourceFormat: String`, provide a regular expression with named capture groups and a `setLabels` map. Output keys are filled from `{{ group_name }}` templates. If multiple `ParseMessage` transformations are used, the one that parses the string must be applied last.

You cannot set `parseMessage.targetLabel` to `.parsed_data`.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: string-to-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: String
        string:
          regex: "^(?P<msg>.*)$"
          setLabels:
            msg: "{{ msg }}"
```

Example original log entry:

```
/docker-entrypoint.sh: Configuration complete; ready for start up
```

Transformed result:

```
{... "message": {
  "msg": "/docker-entrypoint.sh: Configuration complete; ready for start up"
  }
}
```

Transforming logs in Klog format into a structured object

You can use the `ParseMessage` transformation to parse logs in Klog format and convert them into a structured object.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: klog-to-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: Klog
```

Example original log entry:

```
I0505 17:59:40.692994   28133 klog.go:70] hello from klog
```

Transformed result:

```
{... "message": {
  "file":"klog.go",
  "id":28133,
  "level":"info",
  "line":70,
  "message":"hello from klog",
  "timestamp":"2025-05-05T17:59:40.692994Z"
  }
}
```

Transforming logs in Syslog format into a structured object

You can use the `ParseMessage` transformation to parse logs in Syslog format and convert them into a structured object.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: syslog-to-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: SysLog
```

Example original log entry:

```
<13>1 2020-03-13T20:45:38.119Z dynamicwireless.name non 2426 ID931 [exampleSDID@32473 iut="3" eventSource= "Application"
eventID="1011"] Try to override the THX port, maybe it will reboot the neural interface!
```

Transformed result:

```
{... "message": {
  "appname": "non",
  "exampleSDID@32473": {
    "eventID": "1011",
    "eventSource": "Application",
    "iut": "3"
  },
  "facility": "user",
  "hostname": "dynamicwireless.name",
  "message": "Try to override the THX port, maybe it will reboot the neural interface!",
  "msgid": "ID931",
  "procid": 2426,
  "severity": "notice",
  "timestamp": "2020-03-13T20:45:38.119Z",
  "version": 1
  }
}
```

Transforming logs in CLF format into a structured object

You can use the `ParseMessage` transformation to parse logs in CLF format and convert them into a structured object.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: clf-to-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: CLF
```

Example original log entry:

```
127.0.0.1 bob frank [10/Oct/2000:13:55:36 -0700] \"GET /apache_pb.gif HTTP/1.0\" 200 2326
```

Transformed result:

```
{... "message": {
  "host": "127.0.0.1",
  "identity": "bob",
  "message": "GET /apache_pb.gif HTTP/1.0",
  "method": "GET",
  "path": "/apache_pb.gif",
  "protocol": "HTTP/1.0",
  "size": 2326,
  "status": 200,
  "timestamp": "2000-10-10T20:55:36Z",
  "user": "frank"
  }
}
```

Transforming logs in logfmt format into a structured object

You can use the `ParseMessage` transformation to parse logs in logfmt format and convert them into a structured object.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: logfmt-to-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: Logfmt
```

Example original log entry:

```
@timestamp=\"Sun Jan 10 16:47:39 EST 2021\" level=info msg=\"Stopping all fetchers\" tag#production=stopping_fetchers
id=ConsumerFetcherManager-1382721708341 module=kafka.consumer.ConsumerFetcherManager
```

Transformed result:

```
{... "message": {
  "@timestamp": "Sun Jan 10 16:47:39 EST 2021",
  "id": "ConsumerFetcherManager-1382721708341",
  "level": "info",
  "module": "kafka.consumer.ConsumerFetcherManager",
  "msg": "Stopping all fetchers",
  "tag#production": "stopping_fetchers"
  }
}
```

Parsing JSON and reducing the nesting depth

You can use the `ParseMessage` transformation to parse log entries in JSON format. Using the `depth` parameter, you can control the nesting depth.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: parse-json
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: JSON
        json:
          depth: 1
```

Example original log entry:

```
{"level" : { "severity": "info" },"msg" : "fetching.module.release"}
```

Transformed result:

```
{... "message": {
  "level" : "{ \"severity\": \"info\" }",
  "msg" : "fetching.module.release"
  }
}
```

Example of parsing mixed-format entries into a structured object

The string transformation must be applied last.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: parse-json
spec:
  ...
  transformations:
  - action: ParseMessage
    parseMessage:
      sourceFormat: JSON
  - action: ParseMessage
    parseMessage:
      sourceFormat: Klog
  - action: ParseMessage
    parseMessage:
      sourceFormat: String
      string:
        regex: "^(?P<text>.*)$"
        setLabels:
          text: "{{ text }}"
```

Example original log entry:

```
/docker-entrypoint.sh: Configuration complete; ready for start up
{"level" : { "severity": "info" },"msg" : "fetching.module.release"}
I0505 17:59:40.692994   28133 klog.go:70] hello from klog
```

Transformed result:

```
{... "message": {
  "text": "/docker-entrypoint.sh: Configuration complete; ready for start up"
  }
}
{... "message": {
  "level" : "{ "severity": "info" }",
  "msg" : "fetching.module.release"
  }
}
{... "message": {
  "file":"klog.go",
  "id":28133,
  "level":"info",
  "line":70,
  "message":"hello from klog",
  "timestamp":"2025-05-05T17:59:40.692994Z"
  }
}
```

Replacing labels

You can use the `ReplaceKeys` transformation to replace `source` with `target` in the specified label keys.

{% alert level=”info” %} To apply the `ReplaceKeys` transformation to the `message` field or its nested fields, the log entry must first be parsed into a structured object using the `ParseMessage` transformation. {% endalert %}

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: replace-dot
spec:
  ...
  transformations:
    - action: ReplaceKeys
      replaceKeys:
        source: "."
        target: "_"
        labels:
          - .pod_labels
```

Example original log entry:

```
{"msg" : "fetching.module.release"} # Pod label pod.app=test
```

Transformed result:

```
{... "message": {
  "msg" : "fetching.module.release"
  },
  "pod_labels": {
    "pod_app": "test"
  }
}
```

Replacing substrings in string fields

You can use the `ReplaceValue` transformation to replace matches of a regular expression (`source`) in scalar string fields. The `target` string can be a literal or use `{{ name }}` for named capture groups from `source`.

{% alert level=”info” %} Apply `ParseMessage` first if you need to run replacement on a field inside a structured `message`. {% endalert %}

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: redact-digits
spec:
  ...
  transformations:
    - action: ReplaceValue
      replaceValue:
        source: \d+
        target: REDACTED
        labels:
          - .message
```

Example original log entry:

```
user id 12345 connected from 10.0.0.1
```

Transformed result:

```
{... "message": "user id REDACTED connected from REDACTED"
}
```

Adding labels from literals and templates

You can use the `AddLabels` transformation to add fields using static values or templates such as `{{ .pod_labels.app }}`. Transformations run after applying `extraLabels`. If label names overlap, the value from `AddLabels` has priority.

The `when` field is an optional list of conditions combined with logical AND. The left-hand side of a condition is a path to a field on the log event: a leading dot and dot-separated segments (for example, `.namespace`, `.pod_labels.app`, `.message.level`). Each list item can be one of the following types:

- Value comparison: `.namespace == "production"`. Supported operators are `==`, `!=`, `=~`, and `!=~`.
- Field existence check: Only a path with no operator, for example, `.pod_labels.team` (the field must exist).
- Field absence check: `!.pod_labels.legacy`.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: add-env-from-pod
spec:
  ...
  transformations:
    - action: AddLabels
      addLabels:
        when:
          - '.namespace == "production"'
        setLabels:
          .env: prod
          .source_app: "{{ .pod_labels.app }}"
```

Transformed result for an event in namespace `production` with pod label `app=api`:

```
{..."namespace": "production", "pod_labels": { "app": "api" } "env": "prod", "source_app": "api", ...
}
```

Removing labels

You can use the `DropLabels` transformation to remove specific labels from log records. Each item under `dropLabels.labels` has a `label` field (the path) and optional `keepKeys`. If `keepKeys` is set for an item, `label` must point to an **object**; inside it, every nested key not listed in `keepKeys` is removed (key names without a leading dot). If `keepKeys` is unset for that item, the path in `label` is removed entirely.

{% alert level=”info” %} To apply the `DropLabels` transformation to the `message` field or its nested fields, the log entry must first be parsed into a structured object using the `ParseMessage` transformation. {% endalert %}

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: drop-label
spec:
  ...
  transformations:
    - action: DropLabels
      dropLabels:
        labels:
          - label: .example
```

Example of removing a specific label from a structured message

The following example shows how you can remove a label from a structured JSON message. The `ParseMessage` transformation is applied first to parse the message, followed by `DropLabels` to remove the specified label.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: drop-label
spec:
  ...
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: JSON
    - action: DropLabels
      dropLabels:
        labels:
          - label: .message.example
```

Example original log entry:

```
{"msg" : "fetching.module.release", "example": "test"}
```

Transformed result:

```
{... "message": {
  "msg" : "fetching.module.release"
  }
}
```

Sending logs to Elasticsearch with transformations

The following is an example of a ClusterLogDestination resource for sending logs to Elasticsearch.

In this example:

- JSON from the `message` field is parsed and moved into the event root (`targetLabel: "."`).
- Dots are replaced with underscores in `pod_labels` key names.
- The source `message` field is deleted.

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: es-storage
spec:
  type: Elasticsearch
  elasticsearch:
    endpoint: http://elasticsearch:9200
    index: logs-%F
  transformations:
    - action: ParseMessage
      parseMessage:
        sourceFormat: JSON
        targetLabel: "."
    - action: ReplaceKeys
      replaceKeys:
        source: "."
        target: "_"
        labels:
          - .pod_labels
    - action: DropLabels
      dropLabels:
        labels:
          - label: .message
```

Example source log line: a Java app emits structured JSON to stdout, therefore the JSON line is stored in the `message` field of the event. The `msg` field inside that line holds the program log text.

```
{... "namespace": "prod", "pod_labels": {"app.kubernetes.io/name": "orders", "app": "api"}, "message": "
{\"@timestamp\":\"2025-03-25T10:15:00.123Z\",\"level\":\"INFO\",\"logger\":\"com.example.OrderService\",\"msg\":\"Order id=42
created; user=1001\"}"}
```

Transformation result:

- JSON fields are moved into the event root.
- Dots in `pod_labels` key names are replaced with underscores.
- The `message` field is removed.

```
{... "namespace": "prod",
  "pod_labels": {
    "app_kubernetes_io/name": "orders",
    "app": "api"
  },
  "@timestamp": "2025-03-25T10:15:00.123Z",
  "level": "INFO",
  "logger": "com.example.OrderService",
  "msg": "Order id=42 created; user=1001"
}
```

Collect logs from production namespaces using the namespace label selector option

```
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: production-logs
spec:
  type: KubernetesPods
  kubernetesPods:
    namespaceSelector:
      labelSelector:
        matchLabels:
          environment: production
  destinationRefs:
  - loki-storage
```

Exclude Pods or namespaces with a label

There is a preconfigured label to exclude particular namespaces or Pods: `log-shipper.deckhouse.io/exclude=true`. It can help to stop collecting logs from a namespace or Pod without changing global configurations.

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    log-shipper.deckhouse.io/exclude: "true"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-deployment
spec:
  ...
  template:
    metadata:
      labels:
        log-shipper.deckhouse.io/exclude: "true"
```

Enable Buffering

The log buffering configuration is essential for improving the reliability and performance of the log collection system. Buffering can be useful in the following cases:

1. Temporary connectivity disruptions. If there are temporary disruptions or instability in the connection to the log storage system (such as Elasticsearch), a buffer allows logs to be temporarily stored and sent when the connection is restored.
2. Smoothing out load peaks. During sudden spikes in log volume, a buffer helps smooth out peak loads on the log storage system, preventing it from becoming overloaded and potentially losing data.
3. Performance optimization. Buffering helps optimize the performance of the log collection system by accumulating logs and sending them in batches, which reduces the number of network requests and improves overall throughput.

Example of enabling in-memory buffering

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    memory:
      maxEvents: 4096
    type: Memory
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

Example of enabling disk buffering

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    disk:
      maxSize: 1Gi
    type: Disk
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

Example of defining behavior when the buffer is full

```
apiVersion: deckhouse.io/v1alpha1
kind: ClusterLogDestination
metadata:
  name: loki-storage
spec:
  buffer:
    disk:
      maxSize: 1Gi
    type: Disk
    whenFull: DropNewest
  type: Loki
  loki:
    endpoint: http://loki.loki:3100
```

More detailed description of the parameters is available in the ClusterLogDestination resource.

The loki module

⚠ The module lifecycle stage: Preview

In Kubernetes, the system logs on the nodes do not last long and may be lost during a restart or update. This module deploys its own operational log storage based on Grafana Loki in the cluster..

Module features:

- system logs are automatically logged into Loki without additional configuration.;
- access to the logs is implemented via Grafana and the Deckhouse Kubernetes Platform web interface (console);
- the module is designed to store logs for a short time. For long-term storage or archiving, it is recommended to use external systems supported via log-shipper.

Third-party components

List of third-party software used in the `loki` module:

- Loki

  License: GNU Affero General Public License v3.0

  Loki: like Prometheus, but for logs.

Configuration

⚠ The module lifecycle stage: Preview

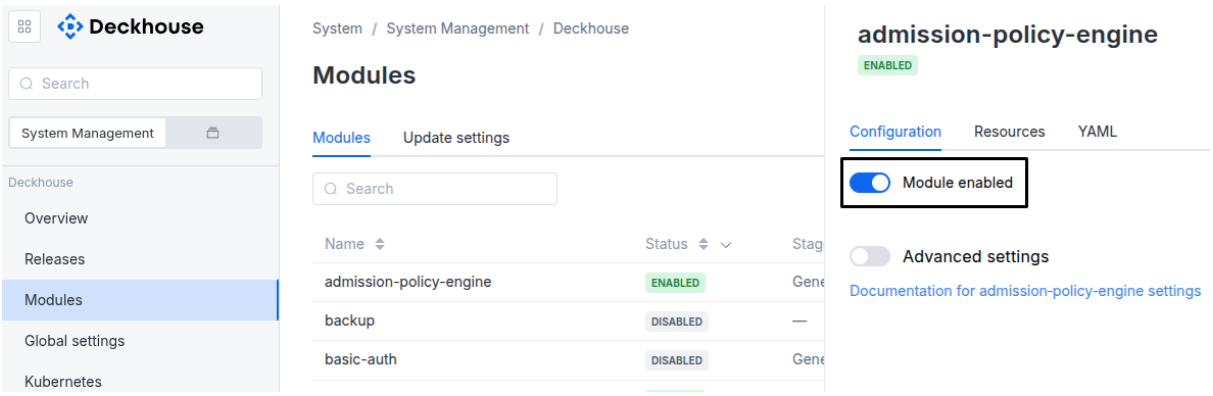
The module has 2 alerts.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `loki` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable loki
```

- **Using ModuleConfig `loki`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `loki` (create it if necessary);

  Example of a manifest to enable module `loki`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: loki
spec:
  enabled: true
```

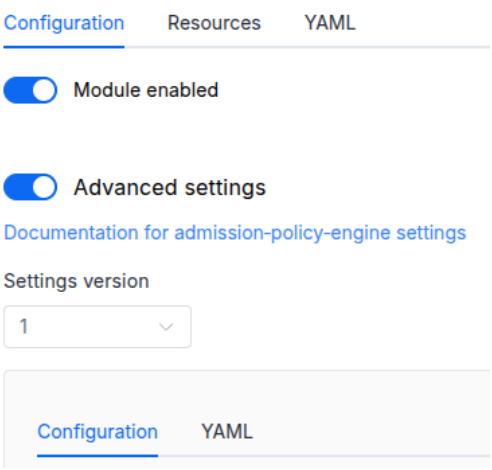
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `loki` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `loki` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `loki`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `loki` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc loki
```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `loki` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

  Example of a manifest for ModuleConfig `loki`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: loki
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.allowDeleteLogs**
    boolean

    Allow logs deletion using /loki/api/v1/delete endpoints.

    Default: `false`
  - **settings.diskSizeGigabytes**
    integer

    Initial disk size for log storage or PVC size when specifying the storageClass parameters. You should manually specify the required disk size for storing logs for the period defined by the `retentionPeriodHours` parameter. The extended-monitoring module automatically monitors the percentage of used disk space.

    Default: `2`
  - **settings.lokiConfig**
    object

    Loki configuration options available for tuning.

    See Loki documentation for more information.

    > **i** Caution! Incorrect settings may lead to the malfunction of Loki.

    - **settings.lokiConfig.grpcServerMaxConcurrentStreams**
      integer

      Limit on the number of concurrent streams for gRPC calls.

      Default: `100`
    - **settings.lokiConfig.grpcServerMaxRecvMsgSize**
      integer

      Limit on the size of a gRPC message this server can receive in bytes.

      Default: `104857600`
    - **settings.lokiConfig.grpcServerMaxSendMsgSize**
      integer

      Limit on the size of a gRPC message this server can send in bytes.

      Default: `4194304`
    - **settings.lokiConfig.ingestionBurstSizeMB**
      number

      Overall ingestion burst size in megabytes. E.g., maximum push request size.

      Default: `6`
    - **settings.lokiConfig.ingestionRateMB**
      number

      Overall ingestion rate limit in megabytes per second.

      Default: `4`
    - **settings.lokiConfig.maxChunksPerQuery**
      integer

      The maximum number of chunks that can be fetched per query.

      Default: `2000000`
    - **settings.lokiConfig.maxEntriesLimitPerQuery**
      integer

      The maximum number of entries returned for the query.

      Default: `5000`
    - **settings.lokiConfig.maxLabelNamesPerSeries**
      integer

      The maximum number of unique label names per series (log stream).

      Default: `32`
    - **settings.lokiConfig.maxStreamsPerUser**
      integer

      The maximum total number of streams for the Loki instance.

      Each stream has unique labels. The number of streams is the number of unique label sets. E. g. `{job="my-app", instance="kube-node-1", namespace="production", pod_name="backend-79dbf6fcb7-v5gs7",`

`app="backend"}`.

0 means unlimited.

Default: `0`

- **settings.lokiConfig.perStreamRateLimit**
  string or number

  Maximum byte rate per second per stream. Can be either a number or a string with a unit (KB, MB, GB).

  Default: `3MB`

  Pattern: `^[0-9]+(KB|MB|GB)?$`

- **settings.lokiConfig.perStreamRateLimitBurst**
  string or number

  Maximum burst size per stream. Can be either a number or a string with a unit (KB, MB, GB).

  Default: `15MB`

  Pattern: `^[0-9]+(KB|MB|GB)?$`

- **settings.nodeSelector**
  object

  The same as the Pods' `spec.nodeSelector` parameter in Kubernetes. If the parameter is omitted or `false`, `nodeSelector` will be determined automatically.

- **settings.resourcesManagement**

  Settings for CPU and memory requests and limits by Loki pods.

  Examples:

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: InPlaceOrRecreate
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

```
resourcesManagement:
  mode: Static
  static:
    requests:
      cpu: 55m
      memory: 256Mi
    limits:
      cpu: 2
      memory: 2Gi
```

```
resourcesManagement:
  mode: VPA
  vpa:
    mode: Auto
    cpu:
      min: 50m
      max: 2
      limitRatio: 1.5
    memory:
      min: 256Mi
      max: 2Gi
      limitRatio: 1.5
```

  - **settings.resourcesManagement.mode**
    string

    Resource management mode:

    - `Static` is a classic one. In it, you specify requests/limits. The parameters of this mode are defined in the static parameter section;
    - `VPA` mode uses VPA. You can configure this mode by modifying parameters in the vpa parameter section.

    Default: `VPA`

Allowed values: `VPA`, `Static`

- **settings.resourcesManagement.static**
  object

  Resource management options for the `Static` mode.

  - **settings.resourcesManagement.static.limits**
    object

    Configuring CPU and memory limits.

    - **settings.resourcesManagement.static.limits.cpu**
      string or number

      Configuring CPU limits.

      Pattern: `^[0-9]+m?$`

    - **settings.resourcesManagement.static.limits.memory**
      string or number

      Configuring memory limits.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  - **settings.resourcesManagement.static.requests**
    object

    Resource requests settings for pods.

    - **settings.resourcesManagement.static.requests.cpu**
      string or number

      Configuring CPU requests.

      Pattern: `^[0-9]+m?$`

    - **settings.resourcesManagement.static.requests.memory**
      string or number

      Configuring memory requests.

      Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.vpa**
  object

  Resource management options for the `VPA` mode.

  - **settings.resourcesManagement.vpa.cpu**
    object

    CPU-related VPA settings.

    - **settings.resourcesManagement.vpa.cpu.limitRatio**
      number

      The CPU limits/requests ratio.

      This ratio is used for calculating the initial CPU limits for a pod.

      If this parameter is set, the VPA will recalculate the CPU limits while maintaining the specified limits/requests ratio.

    - **settings.resourcesManagement.vpa.cpu.max**
      string or number

      The maximum value that the VPA can set for the CPU requests.

      Default: `2`

      Pattern: `^[0-9]+m?$`

    - **settings.resourcesManagement.vpa.cpu.min**
      string or number

      The minimum value that the VPA can set for the CPU requests.

      Default: `50m`

      Pattern: `^[0-9]+m?$`

  - **settings.resourcesManagement.vpa.memory**
    object

    Memory-related VPA settings.

    - **settings.resourcesManagement.vpa.memory.limitRatio**
      number

      The memory limits/requests ratio.

      This ratio is used for calculating the initial memory limits for a pod.

If this parameter is set, the VPA will recalculate the memory limits while maintaining the specified limits/requests ratio.

- **settings.resourcesManagement.vpa.memory.max**
  string or number

  The maximum memory requests the VPA can set.

  Default: `2Gi`

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.vpa.memory.min**
  string or number

  The minimum memory requests the VPA can set.

  Default: `256Mi`

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **settings.resourcesManagement.vpa.mode**
  string

  VPA operating mode.

  - `Initial` — VPA sets initial values for pod resource requests and limits at pod creation time. Resource values are not changed automatically afterwards.
  - `InPlaceOrRecreate` — VPA attempts to update pod resources in place when supported by the cluster. If in-place updates are not possible, the pod will be recreated.
  - `Auto` — VPA automatically recreates pods to apply updated resource values.

    > **i** Starting from Deckhouse version 1.75, this mode is considered legacy. It is recommended to use `InPlaceOrRecreate` instead.

  Default: `InPlaceOrRecreate`

  Allowed values: `Initial`, `InPlaceOrRecreate`, `Auto`

- **settings.retentionPeriodHours**
  integer

  The desired minimum log retention period in hours.

  This parameter does not affect the log retention depth, since Loki only deletes old data when the disk is full.

  If Loki deletes data that has not yet reached the specified retention period, you will receive a `LokiInsufficientDiskForRetention` alert. In this case, you should either increase the disk size or decrease the value of this parameter.

  Default: `168`

- **settings.storageClass**
  string or boolean

  The name of the StorageClass for storing Loki logs (PVC).

  If omitted, the StorageClass of the existing Loki PVC is used. If there is no PVC yet, the StorageClass will be used according to the global storageClass parameter setting.

  The global `storageClass` parameter is only considered when the module is enabled. Changing the global `storageClass` parameter while the module is enabled will not trigger disk re-provisioning.

  Warning. Specifying a value different from the one currently used (in the existing PVC) will result in disk re-provisioning and all data will be deleted.

  Allowed values: `false`

  Examples:

  ```
  storageClass: false
  ```

  ```
  storageClass: default
  ```

- **settings.storeSystemLogs**
  boolean

  Save logs from the `d8-*` namespaces to loki.

  The log-shipper module must be enabled.

  Default: `true`

- **settings.tolerations**
  array of objects

  The same as the Pods' `spec.tolerations` parameter in Kubernetes. If the parameter is omitted or `false`, `tolerations` will be determined automatically.

- **settings.tolerations.effect**
  string
- **settings.tolerations.key**
  string
- **settings.tolerations.operator**
  string
- **settings.tolerations.tolerationSeconds**
  integer
- **settings.tolerations.value**
  string

An example of the configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: loki
spec:
  settings:
    storageClass: ceph-csi-rbd
    diskSizeGigabytes: 10
    retentionPeriodHours: 48
  enabled: true
  version: 1
```

FAQ

⚠ The module lifecycle stage: Preview

Why is LokiDiscardedSamplesWarning alert triggered?

The `LokiDiscardedSamplesWarning` alert is triggered when Loki discards logs during data ingestion because they fail validation or exceed configured limits.

In other words, `log-shipper` is trying to send data that Loki rejects.

Possible causes

Most likely, a limit on the size or number of streams has been exceeded.

Loki enforces limits on:

- The number of unique labels per stream
- The length of label names and values
- The number of streams per client
- The batch size during ingestion

For a detailed description of each limit, refer to the official Loki documentation.

Resolving the issue

1. Check the reason in the alert description.

   ```
   Samples are being discarded because of "{{ $labels.reason }}"...
   ```

   The `{{ $labels.reason }}` field indicates which limit was exceeded and why Loki is rejecting logs.

2. Adjust the corresponding limit in the `loki` module configuration.

   For a description of configurable parameters, refer to the module documentation.

   For example, if the reason is `rate_limited`, increase the `ingestionRateMB` or `ingestionBurstSizeMB` values if temporary ingestion spikes are expected. Alternatively, you can limit the amount of logs sent on the application side.

Module loki: examples

⚠ The module lifecycle stage: Preview

Reading Pod logs from a specified namespace and sending them to Loki

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: loki
spec:
  settings:
    storageClass: ceph-csi-rbd
    diskSizeGigabytes: 30
    retentionPeriodHours: 168
  enabled: true
  version: 1
---
apiVersion: deckhouse.io/v1alpha2
kind: ClusterLoggingConfig
metadata:
  name: development-logs
spec:
  type: KubernetesPods
  kubernetesPods:
    namespaceSelector:
      labelSelector:
        matchExpressions:
        - key: "kubernetes.io/metadata.name"
          operator: In
          values: [development]
  destinationRefs:
    - d8-loki
```

For more examples, see the log-shipper module page.

The metallb module

> **i** Available with limitations in: **SE, SE+, CSE Lite (1.73)**
>
> Available without limitations in: **EE**

> **i** The module lifecycle stage: General Availability

This module implements the `LoadBalancer` mechanism for services in bare metal clusters.

Supports the following operating modes:

- **Layer 2 Mode** – introduces an improved load-balancing mechanism for bare metal clusters (compared to the standard L2 mode in MetalLB), enabling the use of multiple “public” IP addresses for cluster services.
- **BGP Mode** – fully based on the MetalLB solution.

Layer 2 mode

In Layer 2 mode, one or more nodes take responsibility for providing the service within the “public” network. From the network’s perspective, it appears as if each of these nodes has multiple IP addresses assigned to its network interface. Technically, this is achieved by the module responding to ARP requests for IPv4 services and NDP requests for IPv6 services. The primary advantage of Layer 2 mode is its versatility: it works in any Ethernet network without requiring specialized hardware.

Advantages of the module over the classic MetalLB

MetalLB in L2 mode allows ordering Service with `LoadBalancer` type, the operation of which is based on the fact that balancing nodes simulate ARP-responses from the “public” IP in a peering network. This mode has a significant limitation — only one balancing node handles all the incoming traffic of this service at a time. Therefore:

- The node selected as the leader for the “public” IP becomes a “bottleneck”, with no possibility of horizontal scaling.
- If the balancer node fails, all current connections will be dropped while switching to a new balancing node that will be selected as the leader.

This module helps to overcome these limitations. It introduces a new resource, MetalLoadBalancerClass, which allows associating a group of nodes with an IP address pool using a `nodeSelector`. Afterward, a standard Service resource of type `LoadBalancer` can be created, specifying the name of the corresponding MetalLoadBalancerClass. Additionally, annotations can be used to define the required number of IP addresses for L2 advertisement.

Thus:

- The application will receive not a single, but several (according to the number of balancer nodes) “public” IPs. These IPs will need to be configured as A-records for the application’s public domain. For further horizontal scaling, additional balancer nodes will need to be added, the corresponding Service will be created automatically, you just need to add them to the list of A-records for the application domain.
- If one of the balancer nodes fails, only part of the connections will fail over to the healthy node.

BGP Mode

Metallb in BGP mode provides an efficient and scalable way to expose `LoadBalancer` type Services in Kubernetes clusters running on bare metal. By utilizing the standardized BGP protocol, metallb seamlessly integrates into existing network infrastructure and ensures high availability of Services.

How metallb Works in BGP Mode

In BGP mode, metallb establishes BGP sessions with routers (or Top-of-Rack switches) and announces the IP addresses of `LoadBalancer` type services to them. This is accomplished as follows:

Configuration.

- Metallb is configured with a pool of IP addresses that it can assign to Services.
- BGP session parameters are defined: the Autonomous System (AS) number of the Kubernetes cluster, the IP addresses of the routers (peers), the AS number of the peers, and optionally, authentication passwords.
- For each IP address pool, specific announcement parameters can be set, such as community strings.

Establishing BGP Sessions.

- On each node of the Kubernetes cluster where metallb is running, the speaker component establishes BGP sessions with the specified routers.
- Routing information is exchanged between metallb and the routers.

Assigning IP Addresses to Services.

- When a Service of type `LoadBalancer` is created, metallb selects a free IP address from the configured pool and assigns it to the Service.
- The controller component tracks changes to Services and manages IP address assignments.

Announcing IP Addresses.

- After an IP address is assigned, the speaker on the node elected as the leader for that Service begins announcing the IP address over the established BGP sessions.
- The routers receive this announcement and update their routing tables, directing traffic for that IP address to the node that announced it.

Traffic Distribution.

- Routers use Equal-Cost Multi-Path (ECMP) or other load balancing algorithms to distribute traffic among nodes announcing the same Service IP address.
- Inside the Kubernetes cluster, traffic arriving at a node is forwarded to the Service’s pods using the mechanisms of the employed CNI (iptables/IPVS, eBPF programs, etc.).

Advantages of Using BGP

- **Standardized Protocol:** BGP is a widely used and well-established routing protocol.
- **Flexibility and Scalability:** BGP allows you to integrate your Kubernetes cluster into your existing network infrastructure and scale your services.
- **Load Distribution:** Using ECMP allows for efficient load distribution across cluster nodes.
- **Fault Tolerance:** If a node announcing an IP address fails, routers automatically redirect traffic to other nodes announcing the same IP.

Disadvantages of Using BGP

- **Configuration Complexity:** Configuring BGP can be more complex than configuring ARP/NDP announcements (in Layer 2 mode).
- **Network Equipment Requirements:** Routers must support BGP and ECMP.

Third-party components

List of third-party software used in the `metallb` module:

- MetalLB 0.15.2

  License: Apache License 2.0

  A load-balancer implementation for bare metal Kubernetes clusters, using standard routing protocols.

Configuration

> **i** Available with limitations in: **SE, SE+, CSE Lite (1.73)**
>
> Available without limitations in: **EE**

> **i** The module lifecycle stage: General Availability

The module has 7 alerts.

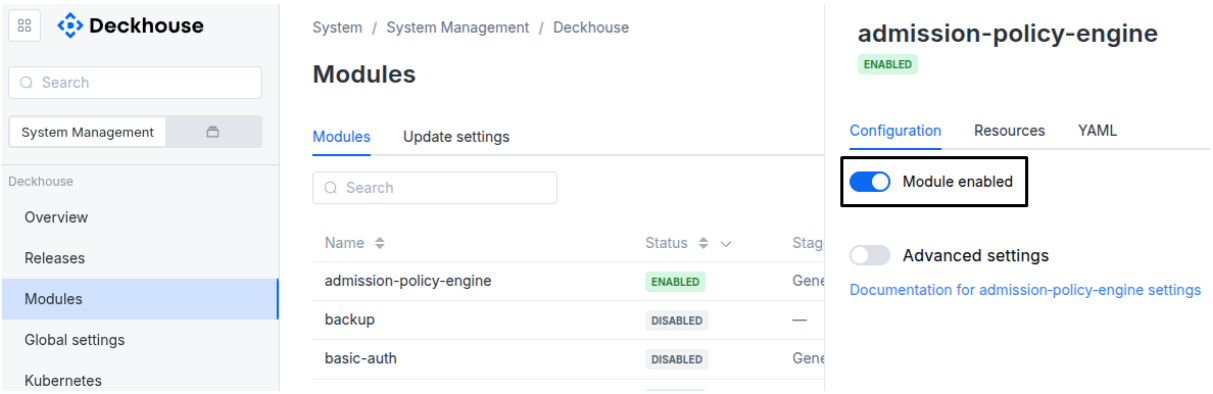
The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the

`metallb` module and enable (or disable) the “Module enabled” toggle. Save changes.

Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable metallb
  ```

- **Using ModuleConfig `metallb`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `metallb` (create it if necessary);

  Example of a manifest to enable module `metallb`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: metallb
  spec:
    enabled: true
  ```

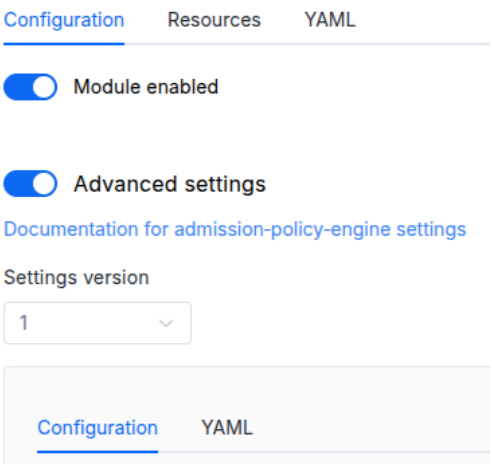
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `metallb` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `metallb` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `metallb`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `metallb` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc metallb
  ```

Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `metallb` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `metallb`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  version: 2
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Remove all `addressPool` elements with the `layer2` protocol.

Parameters

Schema version: 2

Example:

```
bgpPeers:
- peer-address: 192.168.1.1
  peer-asn: 1111
  my-asn: 2222
  source-address: 192.168.1.2
  hold-time: 10s
  node-selector:
    matchLabels:
      node: test
addressPools:
- name: my-pool-bgp
  protocol: bgp
  addresses:
  - 192.168.100.1-192.168.100.10
  - 192.168.101.0/24
  bgp-advertisements:
  - aggregation-length: 32
    localpref: 100
    communities:
    - no-advertise
bgpCommunities:
  no-advertise: 65535:65282
speaker:
  nodeSelector:
    mylabel: speaker
```

- **settings**
  object
  - **settings.addressPools**
    array of objects

    Required value

    A list of IP ranges to assign to services.

    Format — a data array similar to that of MetalLB's.

    Default: `[]`

    - **settings.addressPools.addresses**
      array of strings

Required value

A list of ranges, where each range can look like a subnet/mask or a numeric address range (with “-“ as a delimiter).

- **settings.addressPools.auto-assign**
  boolean

  Auto-assign flag used to prevent metallb from automatic allocation for a pool.

  Default: `true`

- **settings.addressPools.avoid-buggy-ips**
  boolean

  Prevents addresses ending with `.0` and `.255` to be used by a pool.

  Default: `false`

- **settings.addressPools.bgp-advertisements**
  array of objects

  Defines BGP advertisements.

  - **settings.addressPools.bgp-advertisements.aggregation-length**
    integer

    The aggregation-length advertisement option lets you “roll up” prefix into a larger one.

    Works for IPv4 addresses.

    Default: `32`

    Allowed values: `1 <= X`

  - **settings.addressPools.bgp-advertisements.communities**
    array of strings

    Keys from the bgpCommunities parameter to be associated with the announcement.

    Example:

    ```
    communities:
    - no-advertise
    ```

  - **settings.addressPools.bgp-advertisements.localpref**
    integer

    The `BGP LOCAL_PREF` attribute which is used by BGP best path algorithm.

    Path with higher localpref is preferred over one with lower localpref.

- **settings.addressPools.name**
  string

  Required value

  The name of the pool. It should conform to RFC 1123: dot-separated parts in lowercase, consists of alphanumeric characters, ‘-‘. Each part must start and end with an alphanumeric character.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.addressPools.protocol**
  string

  Required value

  The protocol used by the speaker to announce services.

  Allowed values: `bgp`, `layer2`

- **settings.bgpCommunities**
  object

  Available in editions: EE

  The BGP communities list.

  Example:

  ```
  bgpCommunities:
    no-advertise: 65535:65282
  ```

- **settings.bgpPeers**
  array of objects

  Available in editions: EE

  A list of external BGP routers to use with the module.

  Format — a data array similar to that of MetalLB’s.

Default: `[]`

- **settings.bgpPeers.hold-time**
  integer or string

  The timeout after which the neighboring BGP peer is considered dead. This value is divided by three to get the keep-alive interval.

  The recommended value is `3s` (i.e., keep-alive packets are sent once per second). Note that the BGP protocol does not support values lower than this. By default, the parameter is set to `90s` (i.e., keep-alive packets are sent every 30 seconds).

  Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

- **settings.bgpPeers.my-asn**
  integer

  Required value

  The AS number in the cluster.

  Allowed values: `0 <= X <= 4294967295`

- **settings.bgpPeers.node-selector**
  object

  The additional pseudo-selector implemented by the speaker application. It selects nodes that are allowed to connect to external BGP routers. Do not confuse it with `speaker.nodeSelector` and `nodeSelector`.

  An optional parameter.

  The format is `matchLabels` or `matchExpressions`.

  - **settings.bgpPeers.node-selector.matchExpressions**
    array of objects
    - **settings.bgpPeers.node-selector.matchExpressions.key**
      string
    - **settings.bgpPeers.node-selector.matchExpressions.operator**
      string
    - **settings.bgpPeers.node-selector.matchExpressions.values**
      array of strings
  - **settings.bgpPeers.node-selector.matchLabels**
    object
- **settings.bgpPeers.password**
  string

  Authentication password for BGP-routers enforcing TCP MD5 authenticated sessions.

- **settings.bgpPeers.peer-address**
  string

  Required value

  The IP address of the external BGP router.

  Pattern: `^([0-9]{1,3}\.){3}[0-9]{1,3}$`

- **settings.bgpPeers.peer-asn**
  integer

  Required value

  The AS number on the external BGP router.

  Allowed values: `0 <= X <= 4294967295`

- **settings.bgpPeers.peer-port**
  integer

  Port to dial when establishing the session.

  Default: `179`

  Allowed values: `0 <= X <= 16384`

- **settings.bgpPeers.router-id**
  string

  BGP router ID to advertise to the peer.

- **settings.bgpPeers.source-address**
  string

  The source IP address for outbound connections.

  Pattern: `^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$`

- **settings.loadBalancerClass**
  string

  An optional field describing the LoadBalancer class. The LoadBalancerClass field should be used in L2 LoadBalancer mode to specify the MetalLoadBalancerClass that defines the balancer parameters for the Service.

- **settings.nodeSelector**
  object

  A selector for the main controller. It is the same as the Pod's `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

- **settings.speaker**
  object

  Settings for the `speaker` component that implements the LoadBalancer's IPs publishing protocol for external infrastructure.

  - **settings.speaker.nodeSelector**
    object

    Required value

    A selector for the speaker DaemonSet. It is the same as the Pod's `spec.nodeSelector` parameter in Kubernetes.

    If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.speaker.tolerations**
    array of objects

    Tolerations for the speaker DaemonSet. They are the same as the Pod's `spec.tolerations` parameter in Kubernetes.

    If the parameter is omitted or `false`, it will be determined automatically.

    - **settings.speaker.tolerations.effect**
      string
    - **settings.speaker.tolerations.key**
      string
    - **settings.speaker.tolerations.operator**
      string
    - **settings.speaker.tolerations.tolerationSeconds**
      integer
    - **settings.speaker.tolerations.value**
      string

- **settings.tolerations**
  array of objects

  Tolerations for the main controller. They are the same as the Pod's `spec.tolerations` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

MetalLoadBalancerClass

Short names: mlbc

Scope: Cluster
Version: v1alpha1

MetalLoadBalancerClass is the Schema for the metalloadbalancerclasses API

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
- **spec**
  object
  - **spec.addressPool**
    array of strings

An array of public IP address ranges. Each range can be specified using a CIDR notation or by directly specifying the range, for example, `10.10.10.1-10.10.10.100`.

Examples:

```
addressPool: 10.42.0.0/24
```

```
addressPool: 10.33.5.100-10.33.5.106
```

- **spec.isDefault**
  boolean

  Whether MetalLoadBalancerClass is selected for all services by default.

  Example:

  ```
  isDefault: true
  ```

- **spec.l2**
  object

  Available in editions: SE, SE+, EE

  - **spec.l2.interfaces**
    array of strings

    Available in editions: SE, SE+, EE

    A list of network interfaces from which the assigned IP addresses will be announced. If this field is not filled in or an empty array is specified, the announcement will occur from all interfaces on the node. It is highly recommended to configure this parameter to prevent the announcement of addresses from all node interfaces.

    Example:

    ```
    interfaces:
    - eth1
    - eth2
    ```

- **spec.nodeSelector**
  object

  Allows restricting the list of nodes from which the announcement will occur. Specified as a set of labels for filtering nodes.

- **spec.type**
  string

  Selecting the load balancer operating mode.

  Example:

  ```
  type: L2
  ```

- **status**
  object

Examples

> **i** The module lifecycle stage: General Availability

Metallb can be used in Static (bare metal) clusters when there is no option to use cloud load balancers. Metallb can work in L2 LoadBalancer or BGP modes LoadBalancer.

Example of metallb usage in L2 LoadBalancer mode

Enable the module:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  version: 2
```

Prepare the application to publish:

```
d8 k create deploy nginx --image=nginx
```

Create the MetalLoadBalancerClass resource:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: MetalLoadBalancerClass
metadata:
  name: ingress
spec:
  addressPool:
    - 192.168.2.100-192.168.2.150
  isDefault: false
  nodeSelector:
    node-role.kubernetes.io/loadbalancer: "" # node-balancer selector
  type: L2
```

Create standard resource Service with special annotation and MetalLoadBalancerClass name:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  annotations:
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
spec:
  type: LoadBalancer
  loadBalancerClass: ingress # MetalLoadBalancerClass name
  ports:
  - port: 8000
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
```

As a result, the created Service with the type `LoadBalancer` will be assigned the specified number of addresses:

```
d8 k get svc
```

Output example:

```
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP                                 PORT(S)        AGE
nginx-deployment    LoadBalancer   10.222.130.11   192.168.2.100,192.168.2.101,192.168.2.102   80:30544/TCP   11s
```

> **i** When creating a service, you can also specify certain IP addresses from the pool that will be assigned to it.

The resulting EXTERNAL-IP are ready to use in application DNS-domain:

```
$ curl -s -o /dev/null -w "%{http_code}" 192.168.2.100:8000
200
$ curl -s -o /dev/null -w "%{http_code}" 192.168.2.101:8000
200
$ curl -s -o /dev/null -w "%{http_code}" 192.168.2.102:8000
200
```

Example of metallb usage in BGP LoadBalancer mode

Enable the module and configure all the necessary parameters*:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: metallb
spec:
  enabled: true
  settings:
    addressPools:
    - addresses:
      - 192.168.219.100-192.168.219.200
      name: mypool
      protocol: bgp
    bgpPeers:
    - hold-time: 3s
      my-asn: 64600
      peer-address: 172.18.18.10
      peer-asn: 64601
    speaker:
      nodeSelector:
        node-role.deckhouse.io/metallb: ""
  version: 2
```

* — in future versions, BGP mode settings will be set via the MetalLoadBalancerClass resource.

Configure BGP peering on the network equipment.

Additional configuration examples for Service

Creating a service and assigning it specific IP addresses from the pool

> **i** To specify the addresses that should be assigned to the service, use the annotation `network.deckhouse.io/load-balancer-ips`. If there is more than one desired address, there must also be an annotation `network.deckhouse.io/l2-load-balancer-external-ips-count`, which must specify the number of addresses allocated from the pool (it must not be less than the number of addresses listed in `network.deckhouse.io/load-balancer-ips`).

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-deployment
  annotations:
    # The number of addresses that will be allocated from the pool declared in MetalLoadBalancerClass.
    network.deckhouse.io/l2-load-balancer-external-ips-count: "3"
    # A list of addresses from the pool declared in MetalLoadBalancerClass that will be allocated to
the service.
    network.deckhouse.io/load-balancer-ips: "192.168.2.102,192.168.2.103,192.168.2.104"
spec:
  type: LoadBalancer
  loadBalancerClass: ingress # MetalLoadBalancerClass name
  ports:
  - port: 8000
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
```

As a result, the created Service with the type `LoadBalancer` will be assigned the specified number of addresses:

```
d8 k get svc
```

Output example:

```
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP                                 PORT(S)        AGE
nginx-deployment    LoadBalancer   10.222.130.11   192.168.2.102,192.168.2.103,192.168.2.104   80:30544/TCP   11s
```

Creating a service with a single, forcibly selected address

To create a Service with a forcibly selected address, you need to add the annotation `network.deckhouse.io/load-balancer-ips`:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    network.deckhouse.io/load-balancer-ips: 192.168.217.217
spec:
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

Creating services with shared IP addresses

To create a Services with shared IP addresses, you need to add the annotation `network.deckhouse.io/load-balancer-shared-ip-key`. The value of the annotation is a “sharing key”. Services can share an IP address under the following conditions:

- They have the same sharing key.
- They request the use of different ports (e.g. tcp/80 for one and tcp/443 for the other).
- They use the Cluster external traffic policy, or they both point to the exact same set of pods (i.e. the pod selectors are identical).

If these conditions are satisfied, MetalLB may colocate the two services on the same IP, but **does not have to**! If you want to ensure that they share a specific address, use the `network.deckhouse.io/load-balancer-ips` annotation described above.

Here is an example configuration of two services that share the same ip address:

```
apiVersion: v1
kind: Service
metadata:
  name: dns-service-tcp
  namespace: default
  annotations:
    network.deckhouse.io/load-balancer-shared-ip-key: "key-to-share-1.2.3.4"
spec:
  type: LoadBalancer
  ports:
    - name: dnstcp
      protocol: TCP
      port: 53
      targetPort: 53
  selector:
    app: dns
---
apiVersion: v1
kind: Service
metadata:
  name: dns-service-udp
  namespace: default
  annotations:
    network.deckhouse.io/load-balancer-shared-ip-key: "key-to-share-1.2.3.4"
spec:
  type: LoadBalancer
  ports:
    - name: dnsudp
      protocol: UDP
      port: 53
      targetPort: 53
  selector:
    app: dns
```

Creating a service and assigning IPAddressPools to it when using mettalb in BGP LoadBalancer mode

Creating a Service and assigning it IPAddressPools is possible in BGP LoadBalancer mode using the annotation `metallb.universe.tf`. For L2 LoadBalancer mode, you need to use the MetalLoadBalancerClass settings (see above).

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  annotations:
    metallb.universe.tf/address-pool: production-public-ips
spec:
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

Automatic configuration of the monitoring system to collect metrics from user applications

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

Additional configuration is required to monitor user applications running in the cluster. This module simplifies the setup process by reducing it to specifying a specific label for the desired application.

To enable the `monitoring-custom` module to collect application metrics, you must:

- Attach the `prometheus.deckhouse.io/custom-target` label to a Service or a Pod. The label value defines the name in the Prometheus list of targets.
  - You should use the application's name (in lowercase, separated by a hyphen "-") as the value of the `prometheus.deckhouse.io/custom-target` label. This way, the application will be uniquely identified in the cluster.

    Suppose the application is deployed in the cluster more than once (in staging, testing, etc.) or even has several copies in the same namespace. In that case, you still only need to specify its name since all metrics have `namespace`, `pod` labels anyway (and the `service` label if it is accessed via the Service). In other words, the application's name uniquely identifies the application in the cluster (and not a specific installation of it).
- Set the `http-metrics` or `https-metrics` name to the port that will be used for collecting metrics to connect to it over HTTP or HTTPS, respectively.

  If it is not feasible for some reason (e.g., the port is already defined and has a different name), you can use the `prometheus.deckhouse.io/port: port number` annotation to set the port number and `prometheus.deckhouse.io/tls: "true"` if metrics are collected over HTTPS.

  > **i** When annotating a Service, you must use `targetPort` as the port value. I.e., the port that is open and listening by your application, not the Service port.

  - Example No. 1:

    ```
    ports:
    - name: https-metrics
      containerPort: 443
    ```

  - Example No. 2:

    ```
    annotations:
      prometheus.deckhouse.io/port: "443"
      prometheus.deckhouse.io/tls: "true"  # you don't need to specify this annotation if metrics
    are sent over HTTP
    ```
- When using service mesh Istio in mTLS STRICT mode, add the following Service or Pod annotation to force collecting metrics with proper mTLS certificate: `prometheus.deckhouse.io/istio-mtls: "true"`. Note that the application metrics must be exported via pure HTTP without TLS.
- *(Optional)* Attach the following annotations to fine-tune the monitoring:
  - `prometheus.deckhouse.io/path` — the path to collect metrics (default: `/metrics`);
  - `prometheus.deckhouse.io/query-param-$name` — GET parameters; they will be converted to a map of the following form: `$name=$value` ('' by default);
    - you can specify several such annotations;

      For example, `prometheus.deckhouse.io/query-param-foo=bar` & `prometheus.deckhouse.io/query-param-bar=zxc` annotations will be converted to the `http://...?foo=bar&bar=zxc` query;
  - `prometheus.deckhouse.io/allow-unready-pod` — enables collecting metrics for Pods in any state (by default, Prometheus scrapes metrics from the Ready Pods only). Note that this option is rarely helpful. For example, it can come in handy if it takes a long time to start your application (e.g., some data are loaded into the database at

startup or some caching activity occurs), but valuable metrics are supplied during the startup process, helping to monitor it;
- `prometheus.deckhouse.io/sample-limit` — sample limit for a Pod (5000 by default). The default value prevents a situation when the application suddenly starts to supply too many metrics, disrupting the entire monitoring process. This annotation must be attached to the resource with the `prometheus.deckhouse.io/custom-target` label;

An example: Service

```
apiVersion: v1
kind: Service
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    prometheus.deckhouse.io/custom-target: my-app
  annotations:
    prometheus.deckhouse.io/port: "8061"                 # by default, either the http-metrics or
https-metrics service port is used
    prometheus.deckhouse.io/path: "/my_app/metrics"         # /metrics by default
    prometheus.deckhouse.io/query-param-format: "prometheus"  # '' by default
    prometheus.deckhouse.io/allow-unready-pod: "true"       # by default, NON-ready Pods are ignored
    prometheus.deckhouse.io/sample-limit: "5000"           # by default, the sample is limited to 5000
metrics for a single Pod
spec:
  ports:
  - name: my-app
    port: 8060
  - name: http-metrics
    port: 8061
    targetPort: 8061
  selector:
    app: my-app
```

An example: Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
        prometheus.deckhouse.io/custom-target: my-app
      annotations:
        prometheus.deckhouse.io/sample-limit: "5000"  # by default, the sample is limited to 5000
metrics for a single Pod
    spec:
      containers:
      - name: my-app
        image: my-app:1.7.9
        ports:
        - name: https-metrics
          containerPort: 443
```

Configuration

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

The module works if the `prometheus` module is enabled and does not have any parameters to configure.

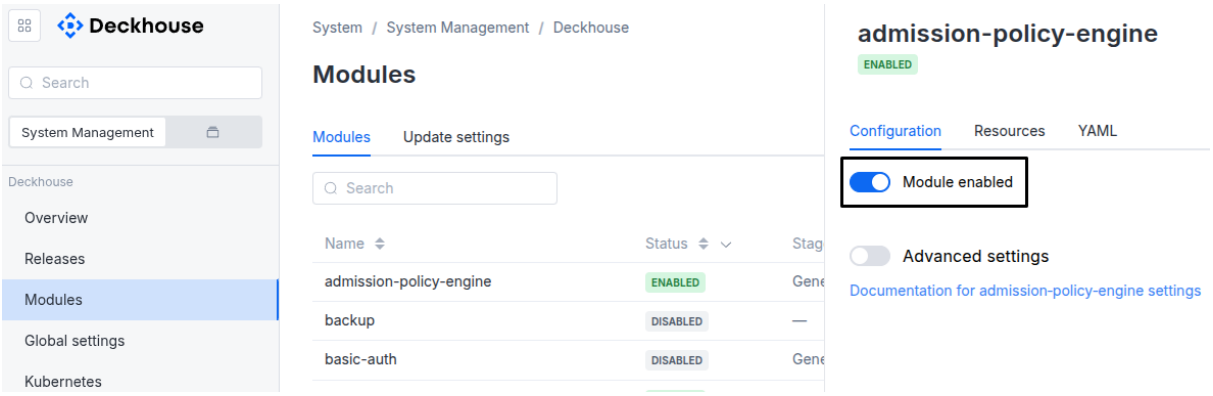
The module has 5 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `monitoring-custom` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable monitoring-custom
```

- **Using ModuleConfig `monitoring-custom`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `monitoring-custom` (create it if necessary);

  Example of a manifest to enable module `monitoring-custom`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-custom
spec:
  enabled: true
```

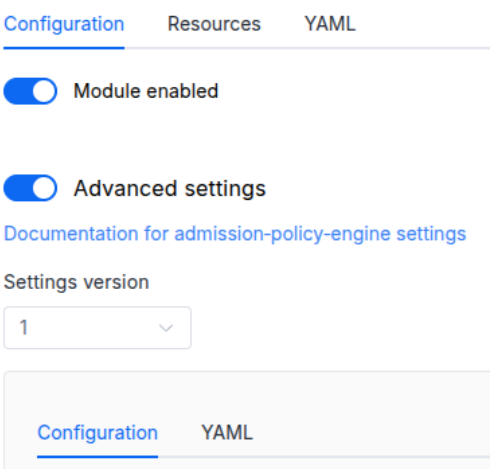
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `monitoring-custom` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `monitoring-custom` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `monitoring-custom`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `monitoring-custom` (for more details on configuring Deckhouse, see the

documentation) by executing the following command:

```
d8 k edit mc monitoring-custom
```

Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `monitoring-custom` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `monitoring-custom`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-custom
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Requirements

To the versions of other modules:

- `prometheus`: any version.

The monitoring-deckhouse module

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

The `monitoring-deckhouse` module provides comprehensive monitoring, alerting, and observability for the Deckhouse Kubernetes Platform itself. It monitors the health, performance, and proper operation of DKP core components to ensure platform stability and reliability.

This module is a critical observability component that works in conjunction with the `prometheus` module to provide insights into DKP operational state.

The module deploys monitoring resources that:

- **Collect Deckhouse metrics**: Module scrapes metrics from the Deckhouse pod using PodMonitor resources, including:
  - Self metrics on port `4222` via `/metrics` endpoint.
  - Custom hook-generated metrics via `/metrics/hooks` endpoint.
  - Module execution metrics, hook performance, and system health indicators.
- **Define alerting rules**: Module provides comprehensive Prometheus alerting rules organized into several categories:
  - **DKP availability**: Monitors pod health, readiness, and uptime.
  - **DKP malfunctioning**: Detects excessive restarts, registry access issues, hung processes.
  - **Release management**: Tracks release channel subscriptions, pending updates, and manual approvals.
  - **Module management**: Monitors module state, validation errors, and deprecated configurations.
  - **CNI checks**: Detects multiple CNI configurations and misconfigurations.
  - **OS requirements**: Identifies nodes running deprecated operating system versions.
- **Provide Grafana dashboards**: Includes pre-built Grafana dashboards for visualizing:
  - DKP performance metrics.
  - Module execution statistics.
  - Hook run times and resource usage.
  - Queue processing and convergence status.

Metrics collection

The module configures a PodMonitor that scrapes two endpoints from the Deckhouse pod:

1. **DKP metrics** (`/metrics`): Core DKP operational metrics:
   - `deckhouse_live_ticks`: Health indicator incrementing every 10 seconds.
   - `deckhouse_registry_errors`: Registry connectivity issues.
   - `deckhouse_module_hook_run_seconds`: Module hook execution duration.
   - `deckhouse_tasks_queue_action_duration_seconds`: Task queue processing times.
   - And many more operational metrics.
2. **Hook metrics** (`/metrics/hooks`): Custom metrics generated by DKP hooks with `honorLabels: true` to preserve hook-specific labels.

Integration with Observability module

When the `observability` module is enabled, this module automatically creates:

- ClusterObservabilityMetricsRulesGroup resources for Prometheus rules.

- ClusterObservabilityDashboard resources for Grafana dashboards.

This enables centralized management and multi-tenancy support for monitoring resources.

Requirements

- `prometheus` module must be enabled (automatic dependency).
- `operator-prometheus` module should be enabled for PodMonitor support.

Configuration

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

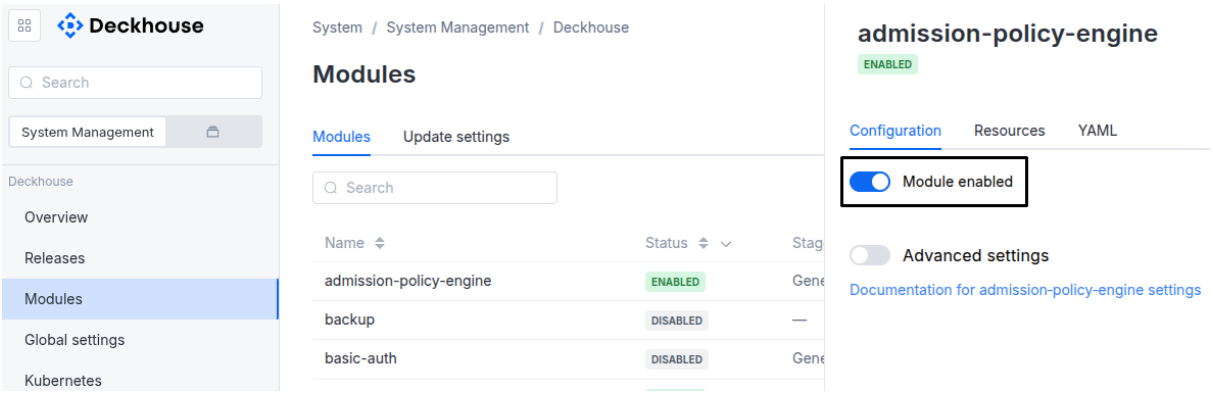
The module has 51 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `monitoring-deckhouse` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable monitoring-deckhouse
```

- **Using ModuleConfig `monitoring-deckhouse`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `monitoring-deckhouse` (create it if necessary);

  Example of a manifest to enable module `monitoring-deckhouse`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-deckhouse
spec:
  enabled: true
```

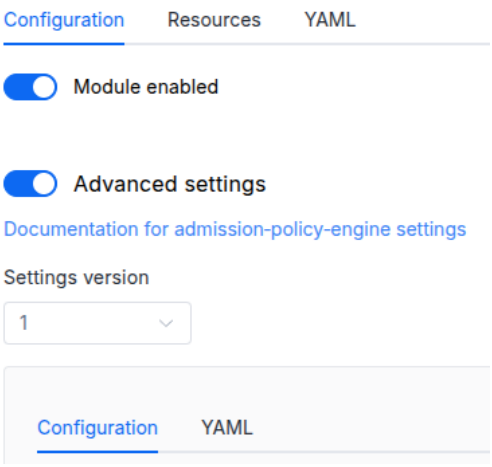
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `monitoring-deckhouse` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `monitoring-deckhouse` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `monitoring-deckhouse`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `monitoring-deckhouse` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc monitoring-deckhouse
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `monitoring-deckhouse` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `monitoring-deckhouse`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: monitoring-deckhouse
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the versions of other modules:

- `prometheus`: any version.

The module does not have any settings.

The monitoring-kubernetes module

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

The `monitoring-kubernetes` module provides transparent and timely monitoring of the status of all cluster nodes and key infrastructure components.

Module features:

- provides an opportunity to plan infrastructure resources (Capacity planning);
- monitors the container runtime version (docker, containerd) on each node and checks it for compliance with the allowed versions;
- monitors the performance of the cluster monitoring subsystem itself (Dead man’s switch);
- gets metrics about the availability of file descriptors, sockets, free space, and inodes on each node;
- monitors the correct operation of key monitoring components: kube-state-metrics, node-exporter, kube-dns;
- checks the status of all nodes (`NotReady`, `drain`, `cordon`) and promptly reports problems;
- monitors time synchronization and notifies about deviations;
- detects cases of prolonged CPU steal overrun (when the node does not receive the required CPU time);

- controls the status of the Conntrack table on the nodes;
- shows pods with incorrect statuses, for example, if kubelet failed to do its job;
- allows you to export metrics to external monitoring systems for a single point of control.

Third-party components

List of third-party software used in the `monitoring-kubernetes` module:

- Node exporter

  License: Apache License 2.0

  Prometheus exporter for hardware and OS metrics exposed by *NIX kernels, written in Go with pluggable metric collectors.

- kube-state-metrics

  License: Apache License 2.0

  A simple service that listens to the Kubernetes API server and generates metrics about the state of the objects.

Configuration

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

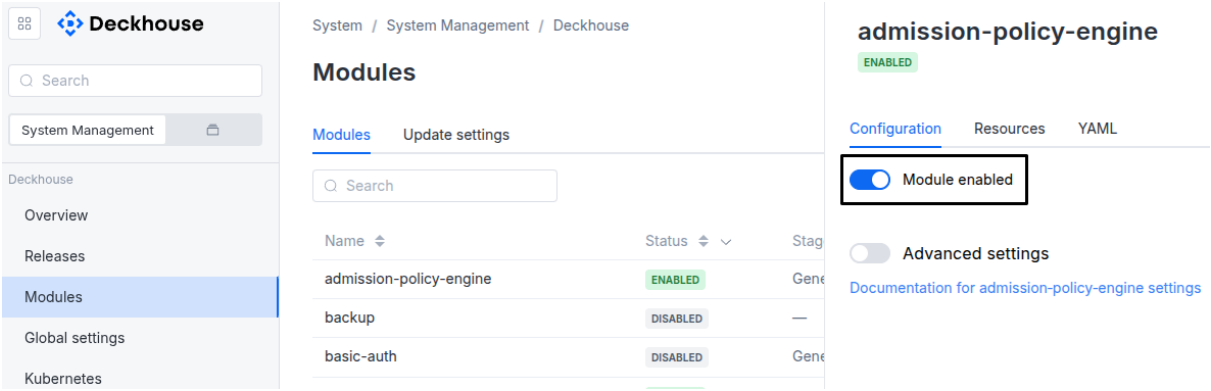
The module has 47 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `monitoring-kubernetes` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable monitoring-kubernetes
```

- **Using ModuleConfig `monitoring-kubernetes`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `monitoring-kubernetes` (create it if necessary);

  Example of a manifest to enable module `monitoring-kubernetes`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-kubernetes
spec:
  enabled: true
```

How to configure the module...

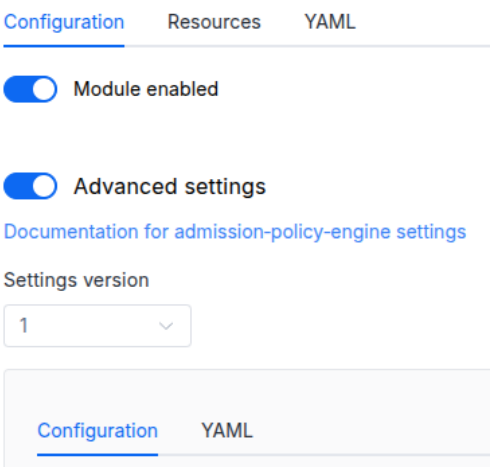
You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `monitoring-kubernetes` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the

module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

Example:



You can also edit the ModuleConfig object `monitoring-kubernetes` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `monitoring-kubernetes`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `monitoring-kubernetes` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc monitoring-kubernetes
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `monitoring-kubernetes` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `monitoring-kubernetes`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: monitoring-kubernetes
  spec:
    version: 2
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the versions of other modules:

- `prometheus`: any version.

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  If the `.ebpfExporterEnabled` field is set to `true`, add `.oomKillsExporterEnabled=true`. Then, delete the `.ebpfExporterEnabled` field from the object.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.highAvailability**
    boolean

Manually enable the high availability mode.

By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

Examples:

```
highAvailability: true
```

```
highAvailability: false
```

- **settings.nodeSelector**
  object

  The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  nodeSelector:
    disktype: ssd
  ```

- **settings.oomKillsExporterEnabled**
  boolean

  Set to 'false' to disable oom-kills-exporter in case it behaves incorrectly.

  Default: `true`

  Example:

  ```
  oomKillsExporterEnabled: false
  ```

- **settings.tolerations**
  array of objects

  The same as in the pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  tolerations:
  - key: key1
    operator: Equal
    value: value1
    effect: NoSchedule
  ```

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

- **settings.vpa**
  object

  VPA settings for `kube-state-metrics` pods.

  Example:

  ```
  vpa:
    kubeStateMetricsMaxCPU: 130m
    kubeStateMetricsMaxMemory: 115Mi
  ```

  - **settings.vpa.kubeStateMetricsMaxCPU**
    string or number

    The maximum value that the VPA can set for the CPU requests.

    Pattern: `^[0-9]+m?$`

    Examples:

    ```
    kubeStateMetricsMaxCPU: 3
    ```

    ```
    kubeStateMetricsMaxCPU: 1000m
    ```

- **settings.vpa.kubeStateMetricsMaxMemory**
  string or number

  The maximum memory requests the VPA can set.

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  Example:

  ```
  kubeStateMetricsMaxMemory: 3Mi
  ```

Monitoring the control plane

> The module lifecycle stage: General Availability
> The module has requirements for installation

The `monitoring-kubernetes-control-plane` module is responsible for monitoring the Kubernetes control plane. It safely scrapes metrics and provides a basic set of rules for monitoring the following cluster components:

- kube-apiserver
- kube-controller-manager
- kube-scheduler
- kube-etcd

Configuration

> The module lifecycle stage: General Availability
> The module has requirements for installation

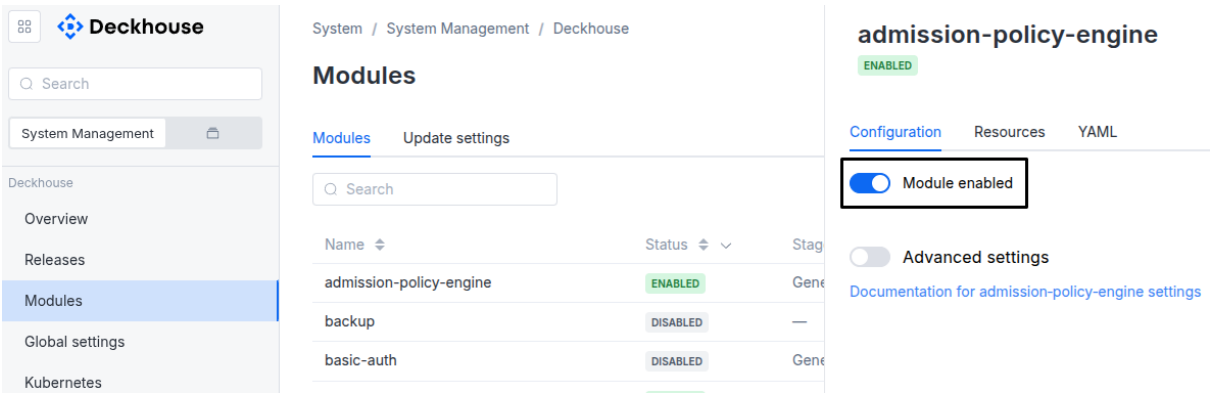
The module has 11 alerts.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `monitoring-kubernetes-control-plane` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:

- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable monitoring-kubernetes-control-plane
  ```

- **Using ModuleConfig `monitoring-kubernetes-control-plane`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `monitoring-kubernetes-control-plane` (create it if necessary);

  Example of a manifest to enable module `monitoring-kubernetes-control-plane`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-kubernetes-control-plane
spec:
  enabled: true
```

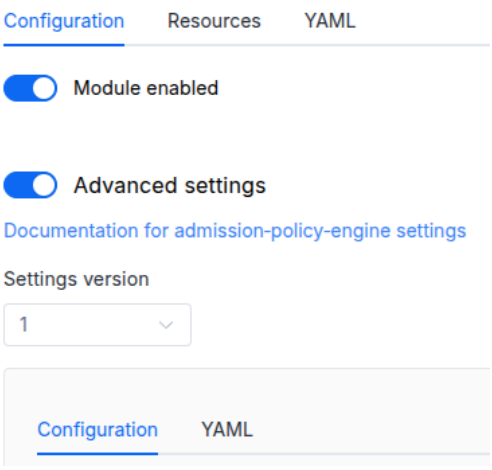
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `monitoring-kubernetes-control-plane` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `monitoring-kubernetes-control-plane` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `monitoring-kubernetes-control-plane`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `monitoring-kubernetes-control-plane` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc monitoring-kubernetes-control-plane
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `monitoring-kubernetes-control-plane` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `monitoring-kubernetes-control-plane`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: monitoring-kubernetes-control-plane
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the Kubernetes version: 1.22.0 and above.

To the Deckhouse version: 1.22.0 and above.

To the versions of other modules:

- `prometheus`: any version.
- `control-plane-manager`: any version.

Monitoring network interaction between all cluster nodes, as well as (optionally) up to additional external nodes

> The module lifecycle stage: General Availability
> The module has requirements for installation

Description

The network communication monitoring module provides continuous connectivity verification between all the main and, if necessary, external nodes of the cluster.

Module features:

- automatically checks the availability of all cluster nodes (and, optionally, external systems) using ICMP (ping) — testing is started every two seconds;
- all results are exported in metrics format to the Prometheus monitoring system.;
- included is a ready—made dashboard for Grafana, where current availability, delay schedules, and potential network connectivity issues are visualized in real time;
- allows you to quickly identify nodes with degraded connectivity and speeds up the response to incidents.

How does it work?

The module tracks the node's `.status.addresses` field for changes. Upon detecting changes, it invokes a hook that collects a complete list of node names/addresses and passes it to a DaemonSet (the latter recreates the Pods). As a result, ping checks the always up-to-date list of nodes.

The monitoring-ping module: configuration

> The module lifecycle stage: General Availability
> The module has requirements for installation

The module does not have any mandatory parameters.

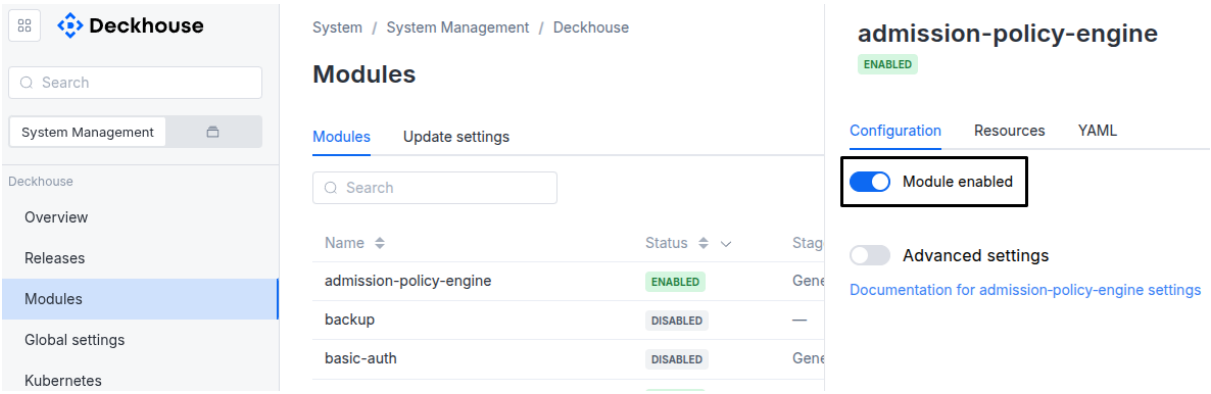
The module has 1 alert.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `monitoring-ping` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable monitoring-ping
```

- **Using ModuleConfig `monitoring-ping`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `monitoring-ping` (create it if necessary);

  Example of a manifest to enable module `monitoring-ping`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-ping
spec:
  enabled: true
```

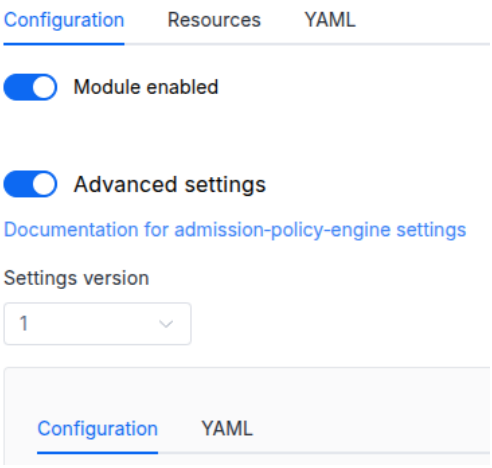
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `monitoring-ping` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `monitoring-ping` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `monitoring-ping`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `monitoring-ping` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc monitoring-ping
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `monitoring-ping` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `monitoring-ping`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: monitoring-ping
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the versions of other modules:

- `monitoring-kubernetes`: any version.

Parameters

Schema version: 1

- **settings**
  object
  - **settings.externalTargets**
    array of objects

    A list of additional hosts to monitor.

    Example:

    ```
    - name: google-primary
      host: 8.8.8.8
    - name: yaru
      host: ya.ru
    - host: youtube.com
    ```

    - **settings.externalTargets.host**
      string

      Required value

      IP address or host name to monitor.

    - **settings.externalTargets.name**
      string

      Monitoring record name.

The monitoring-ping module: usage

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

Adding extra IP addresses to monitor

To monitor additional IP addresses, use the externalTargets parameters:

Module configuration example:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: monitoring-ping
spec:
  version: 1
  enabled: true
  settings:
    externalTargets:
    - name: google-primary
      host: 8.8.8.8
    - name: yaru
      host: ya.ru
    - host: youtube.com
```

> **i** Grafana uses the `name` field to display the related data. If the `name` field is skipped, you must fill in details to the `host` field (mandatory).

Module multitenancy-manager

> ⚠ The module lifecycle stage: Preview

Description

The module enables the creation of projects in a Kubernetes cluster. **Project** is an isolated environment where applications can be deployed.

Why is this needed?

The standard `Namespace` resource, used for logical resource separation in Kubernetes, does not provide necessary functionalities, hence it is not an isolated environment:

- Resource consumption by pods is not limited by default;
- Network communication with other pods works by default from any point in the cluster;
- Unrestricted access to node resources: address space, network space, mounted host directories.

The configuration capabilities of `Namespace` do not fully meet modern development requirements. By default, the following features are not included for `Namespace`:

- Log collection;
- Audit;
- Vulnerability scanning.

The functionality of projects allows addressing these issues.

⚠ The `secret-copier` module cannot be used together with `multitenancy-manager` module.

Advantages of the module

For platform administrators:

- **Consistency**: Administrators can create projects using the same template, ensuring consistency and simplifying management.
- **Security**: Projects provide isolation of resources and access policies between different projects, supporting a secure multitenant environment.
- **Resource Consumption**: Administrators can easily set quotas on resources and limitations for each project, preventing excessive resource usage.

For platform users:

- **Quick Start**: Developers can request projects created from ready-made templates from administrators, allowing for a quick start to developing a new application.
- **Isolation**: Each project provides an isolated environment where developers can deploy and test their applications without impacting other projects.

Limitations

The module works only within the limits below:

- Creating more than one namespace within a project is not supported. If you need multiple namespaces, create a separate project for each of them.
- Template resources are applied only to a single namespace whose name matches the project name.

Internal Logic

Creating a project

To create projects, the following Custom Resources are used:

- ProjectTemplate — a resource that describes the project template. It defines a list of resources to be created in the project and a schema for parameters that can be passed when creating the project;
- Project — a resource that describes a specific project.

When creating a Project resource from a specific ProjectTemplate, the following happens:

1. The parameters passed are validated against the OpenAPI specification (the `parametersSchema.openAPIV3Schema` field of ProjectTemplate);
2. Rendering of the resources template is performed using Helm. Values for rendering are taken from the `parameters` field of the Project resource;
3. A `Namespace` is created with a name matching the name of Project;
4. All resources described in the template are created in sequence.

ℹ **Attention!** When changing the project template, all created projects will be updated according to the new template.

Isolating a project

The project is based on the `Namespace` resource mechanism. Namespaces group pods, services, secrets, and other objects but do not provide complete isolation. The project functionality enhances namespaces by offering additional tools to improve control and security levels. To manage project isolation, Kubernetes features can be leveraged, such as:

- Access control resources (`AuthorizationRule` / `RoleBinding`) — manage interaction with objects within a `Namespace`. Define rules and assign roles to precisely control who can perform actions in your project.
- Resource quotas (`ResourceQuota`) — set limits on resource usage, such as CPU time, RAM, and object counts within a `Namespace`. These quotas help prevent excessive load and maintain control over applications within the project.
- Network connectivity control resources (`NetworkPolicy`) — control incoming and outgoing network traffic within a `Namespace`. Configure allowed connections between pods to enhance security and manage network interactions effectively.

These tools can be combined to configure the project according to the requirements of your application.

The multitenancy-manager module: configuration

⚠ The module lifecycle stage: Preview

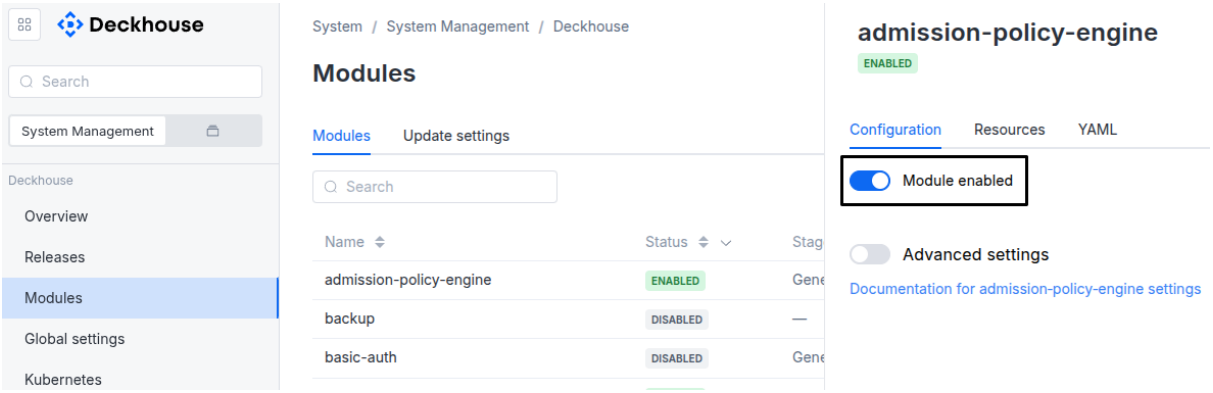
The module does not have any mandatory parameters.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `multitenancy-manager` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable multitenancy-manager
```

- **Using ModuleConfig `multitenancy-manager`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `multitenancy-manager` (create it if necessary);

  Example of a manifest to enable module `multitenancy-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: multitenancy-manager
spec:
  enabled: true
```

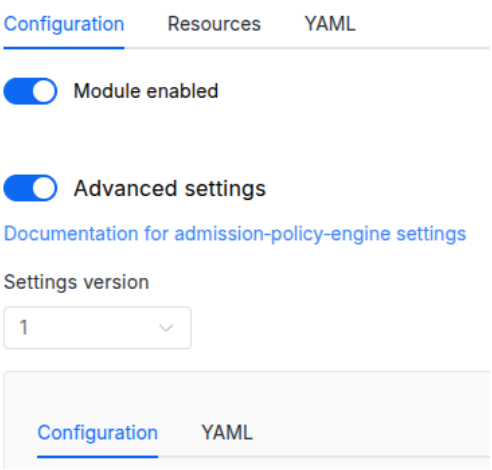
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `multitenancy-manager` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `multitenancy-manager` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `multitenancy-manager`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `multitenancy-manager` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc multitenancy-manager
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `multitenancy-manager` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `multitenancy-manager`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: multitenancy-manager
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.allowNamespacesWithoutProjects**
    boolean

    If this option is disabled - prohibit to create a namespace which is not a part of a Project.

    Default: `true`
  - **settings.highAvailability**
    boolean

    Manually enable the high availability mode.

    By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

    Example:

    ```
    highAvailability: true
    ```

The multitenancy-manager module: Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> ⚠ The module lifecycle stage: Preview

Project

Scope: Cluster

- v1alpha2
- v1alpha1

- **spec**
  object
  - **spec.description**
    string

    Arbitrary description of the project’s purpose. Specify an empty string if no description is required.
  - **spec.parameters**
    object

    Parameters passed to the resources template when creating the project.

Before creation, the values are validated against the schema of the template’s input parameters.

- **spec.projectTemplateName**
  string

  The name of the ProjectTemplate resource that defines which resources will be created in the project.

- **status**
  object
  - **status.conditions**
    array of objects
    - **status.conditions.lastProbeTime**
      string
    - **status.conditions.lastTransitionTime**
      string
    - **status.conditions.message**
      string
    - **status.conditions.status**
      string
    - **status.conditions.type**
      string
  - **status.namespaces**
    array of strings

    Used namespaces.

  - **status.observedGeneration**
    integer

    The last observed project`s generation

  - **status.resources**
    object

    Rendered and skipped resources.

  - **status.state**
    string

    Brief description of the project status, such as Error, Deploying, Deployed, etc.

  - **status.templateGeneration**
    integer

    The last observed template`s generation

Deprecated resource. Support for the resource might be removed in a later release.

- **spec**
  object
  - **spec.description**
    string

    Arbitrary description of the project’s purpose. Specify an empty string if no description is required.

  - **spec.projectTypeName**
    string

    The name of the ProjectType resource that defines which resources will be created in the project.

  - **spec.template**
    object

    Values passed to the resources template when creating the project. Before creation, the values are validated against the schema of the template’s input parameters.

- **status**
  object
  - **status.message**
    string

    Message indicating the cause of the current status.

  - **status.state**
    string

    Brief description of the project status, such as Ready, Error, Pending, etc.

  - **status.sync**
    boolean

    Indicates that the project parameters have been successfully applied after creation or modification.

ProjectTemplate

Scope: Cluster
Version: v1alpha1

Describes the project template.

This resource is referenced by the Project custom resource.

- **spec**
  object
  - **spec.description**
    string

    Arbitrary description of the template's purpose. Specify an empty string if no description is required.

  - **spec.parametersSchema**
    object

    Required value

    One of parameters schema to validate parameters of a project.

    For now only OpenAPI v3 schema is supported.

    - **spec.parametersSchema.openAPIV3Schema**
      object

      Required value

      OpenAPI specification for parameters.

      This is used to check the validity of the parameters that are passed when creating a new project.

  - **spec.resourcesTemplate**
    string

    Required value

    A template that defines resources in the `go template` format to be created when setting up a new project.

    Templates are fully compatible with all `helm` functions.

    Read more about creating an isolated Environment...

    The following parameters are available in templates:

    - `{{ .projectName }}` — the name of the `Project` for which the template is being rendered.
    - `{{ .projectTemplateName }}` — the name of the `ProjectTemplate`.
    - `{{ .parameters }}` — a dictionary of custom values, described in the .spec.parametersSchema and defined in the .spec.parameters.

    > **Note!** Specifying `.metadata.namespace` fields for objects is optional, as this field is automatically set with the name of the created project.

- **status**
  object
  - **status.message**
    string

    Message indicating the cause of the current status.

  - **status.ready**
    boolean

    Whether the template is ready to use. Indicates that the template has been successfully validated.

The multitenancy-manager module: usage examples

> ⚠ The module lifecycle stage: Preview

Default project templates

The following project templates are included in the Deckhouse Kubernetes Platform:

- `default` — a template that covers basic project use cases:
  - resource limitation
  - network isolation
  - automatic alerts and log collection
  - choice of security profile
  - project administrators setup

  Template description on GitHub.

- `secure` — includes all the capabilities of the `default` template and additional features:
  - setting up permissible UID/GID for the project
  - audit rules for project users' access to the Linux kernel
  - scanning of launched container images for CVE presence

  Template description on GitHub.

- `secure-with-dedicated-nodes` — includes all the capabilities of the `secure` template and additional features:
  - defining the node selector for all the pods in the project: if a pod is created, the node selector pod will be **substituted** with the project's node selector automatically.
  - defining the default toleration for all the pods in the project: if a pod is created, the default toleration will be **added** to the pod automatically.

Template description on GitHub.

To list all available parameters for a project template, execute the command:

```
d8 k get projecttemplates <PROJECT_TEMPLATE_NAME> -o jsonpath='{.spec.parametersSchema.openAPIV3Schema}' | jq
```

Creating a project

1. To create a project, create the Project resource by specifying the name of the project template in .spec.projectTemplateName field.
2. In the .spec.parameters field of the Project resource, specify the parameter values suitable for the ProjectTemplate .spec.parametersSchema.openAPIV3Schema.

   Example of creating a project using the Project resource from the `default` ProjectTemplate:

   ```
   apiVersion: deckhouse.io/v1alpha2
   kind: Project
   metadata:
     name: my-project
   spec:
     description: This is an example from the Deckhouse documentation.
     projectTemplateName: default
     parameters:
       resourceQuota:
         requests:
           cpu: 5
           memory: 5Gi
           storage: 1Gi
         limits:
           cpu: 5
           memory: 5Gi
       networkPolicy: Isolated
       podSecurityProfile: Restricted
       extendedMonitoringEnabled: true
       administrators:
       - subject: Group
         name: k8s-admins
   ```

3. To check the status of the project, execute the command:

   ```
   d8 k get projects my-project
   ```

   A successfully created project should be in the `Deployed` state. If the state equals `Error`, add the `-o yaml` argument to the command (e.g., `d8 k get projects my-project -o yaml`) to get more detailed information about the error.

Creating a project automatically for a namespace

You can create a new project for a namespace. To do this, add the `projects.deckhouse.io/adopt` annotation to the namespace. For example:

1. Create a new namespace:

   ```
   d8 k create ns test
   ```

2. Add the annotation:

   ```
   d8 k annotate ns test projects.deckhouse.io/adopt=""
   ```

3. Make sure that the project was created:

   ```
   d8 k get projects
   ```

   A new project corresponding to the namespace will appear in the project list:

   ```
   NAME        STATE      PROJECT TEMPLATE   DESCRIPTION                                          AGE
   deckhouse   Deployed   virtual            This is a virtual project                            181d
   default     Deployed   virtual            This is a virtual project                            181d
   test          Deployed   empty                                                                   1m
   ```

You can change the template of the created project to the existing one.

⚠ Note that changing the template may cause a resource conflict. If the template chart contains resources that are already present in the namespace, you will not be able to apply the template.

Creating your own project template

Default templates cover basic project use cases and serve as a good example of template capabilities.

To create your own template:

1. Take one of the default templates as a basis, for example, `default`.
2. Copy it to a separate file, for example, `my-project-template.yaml` using the command:

   ```
   d8 k get projecttemplates default -o yaml > my-project-template.yaml
   ```

3. Edit the `my-project-template.yaml` file, make the necessary changes.

   {% alert level=”info” %} You must update not only the template itself, but also the input parameters schema to match it.

   Project templates support all Helm templating functions. {% endalert %}

4. Change the template name in the `.metadata.name` field.
5. Apply your new template with the command:

   ```
   d8 k apply -f my-project-template.yaml
   ```

6. Check the availability of the new template with the command:

   ```
   d8 k get projecttemplates <NEW_TEMPLATE_NAME>
   ```

Using labels to manage resources

When creating resources in `ProjectTemplate`, you can use special labels to control how the `multitenancy-manager` processes these resources.

Skipping creation of the `heritage: multitenancy-manager` label

By default, all resources created from `ProjectTemplate` receive the label `heritage: multitenancy-manager`.
This label prohibits changes to resources by users or any other controller except `multitenancy-manager`.
If you need to allow resource modification (for example, for compatibility with other systems, or if implementing your own control over the created objects), add the label `projects.deckhouse.io/skip-heritage-label` to the resource.

Example:

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: {{ .projectName }}
  labels:
    projects.deckhouse.io/skip-heritage-label: "true"
    app: my-app
data:
  key: value
```

In this case, the resource will receive the labels `projects.deckhouse.io/project` and `projects.deckhouse.io/project-template`, but will not receive the label `heritage: multitenancy-manager`.

Excluding resources from management by multitenancy-manager

If you need to exclude a resource from management by `multitenancy-manager` (for example, if the resource should be managed manually or by another controller), add the label `projects.deckhouse.io/unmanaged` to the resource.

Example:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: external-secret
  namespace: {{ .projectName }}
  labels:
    projects.deckhouse.io/unmanaged: "true"
type: Opaque
data:
  token: <base64-encoded-value>
```

Resources with the label `projects.deckhouse.io/unmanaged`:

- Will be created **only once** when the project is created;
- **Will not be updated** with subsequent template changes or updates;
- Will not be monitored in the project's status;
- Will receive the labels `projects.deckhouse.io/project` and `projects.deckhouse.io/project-template` but **will not receive** the label `heritage: multitenancy-manager`.

⚠ Once a resource is marked as `unmanaged`, it will be created on initial installation but not updated when the `ProjectTemplate` is changed.
After creation, the resource becomes fully independent and must be managed manually.

Implementing validation of object changes with a custom label

The `multitenancy-manager` module uses `ValidatingAdmissionPolicy` to protect resources labeled `heritage: multitenancy-manager` from manual changes.
You can implement similar validation for resources with any label.

How validation works in multitenancy-manager

Validation occurs for objects labeled `heritage: multitenancy-manager`.
The following components are used for this:

1. `ValidatingAdmissionPolicy`: Defines validation rules:
   - Operations: `UPDATE` and `DELETE`.
   - Check: only operations on behalf of the controller's service account are allowed.
   - Applies to all resources and API groups.
2. `ValidatingAdmissionPolicyBinding`: Defines which objects the validation applies to:
   - Uses `namespaceSelector` and `objectSelector` to select resources by the label `heritage: multitenancy-manager`.

Creating your own validation

To implement validation for resources with a different label (for example, `heritage: my-custom-label`):

1. Create a file with the ValidatingAdmissionPolicy and ValidatingAdmissionPolicyBinding resource manifests:

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: my-custom-label-validation
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
      - apiGroups:   ["*"]
        apiVersions: ["*"]
        operations:  ["UPDATE", "DELETE"]
        resources:   ["*"]
        scope: "*"
  validations:
    - expression: 'request.userInfo.username == "system:serviceaccount:my-namespace:my-service-
account"' # Replace with your service account
      reason: Forbidden
      messageExpression: 'object.kind == ''Namespace'' ? ''This resource is managed by '' +
object.metadata.name + '' system. Manual modification is forbidden.''
        : ''This resource is managed by '' + object.metadata.namespace + '' system. Manual
modification is forbidden.'''
---
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
  name: my-custom-label-validation
spec:
  policyName: my-custom-label-validation
  validationActions: [Deny, Audit]
  matchResources:
    namespaceSelector:
      matchLabels:
        heritage: my-custom-label
    objectSelector:
      matchLabels:
        heritage: my-custom-label
```

2. Configure the validation parameters:
   - `policyName`: Unique policy name (must match in Policy and Binding).
   - `request.userInfo.username`: The name of the service account allowed to change resources (replace with your

service account).
  - `heritage: my-custom-label`: The value of the `heritage` label for your resources (replace with your value). The use of the values `multitenancy-manager`, `deckhouse` is prohibited.
  - `failurePolicy: Fail`: Policy on validation failure.
    - `Fail`: Reject the request on validation failure.
    - `Ignore`: Ignore validation errors.
  - `validationActions`: Validation actions:
    - `Deny`: Deny unauthorized operations.
    - `Audit`: Record operations in the audit log.

3. Apply the policy:

```
d8 k apply -f my-validation-policy.yaml
```

4. Ensure your resources have the corresponding `heritage` label:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-resource
  labels:
    heritage: my-custom-label
```

The namespace-configurator module

> **i** The module lifecycle stage: General Availability

The `namespace-configurator` module allows to assign annotations and labels to namespaces automatically.

It facilitates to enable new namespaces to monitoring system by adding `extended-monitoring.deckhouse.io/enabled=true` label.

How does it work?

This module monitors the namespaces and configuration:

- It assigns labels and annotations from the configuration to all namespaces that match the `includeNames` pattern and do not match the `excludeNames` pattern.
- Namespaces labeled `heritage` with one of the values `upmeter`, `deckhouse` or `multitenancy-manager` are ignored;
- When changing the module configuration, namespace labels and annotations will be reassigned according to the configuration.

What do I need to configure?

All you need to do is to specify list of desired labels and annotations and matching patterns for namespaces in the module configuration.

Configuration

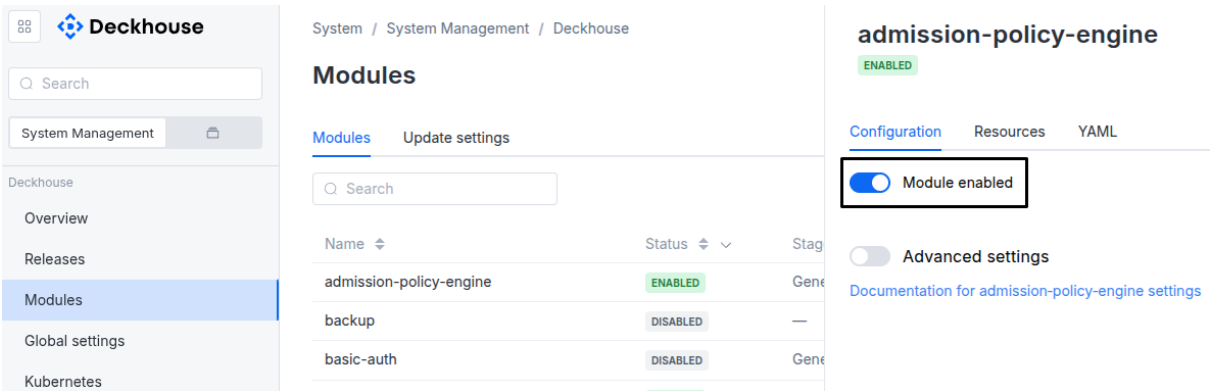
> **i** The module lifecycle stage: General Availability

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `namespace-configurator` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

Example of enabling the module:

```
d8 system module enable namespace-configurator
```

- **Using ModuleConfig `namespace-configurator`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `namespace-configurator` (create it if necessary);

  Example of a manifest to enable module `namespace-configurator`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: namespace-configurator
spec:
  enabled: true
```

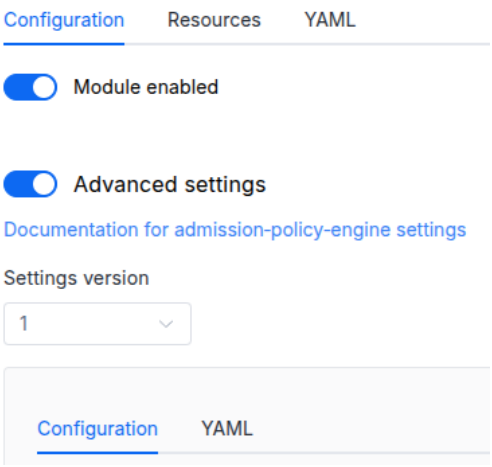
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `namespace-configurator` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `namespace-configurator` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `namespace-configurator`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `namespace-configurator` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc namespace-configurator
```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `namespace-configurator` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

  Example of a manifest for ModuleConfig `namespace-configurator`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: namespace-configurator
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.configurations**
    array of objects
    - **settings.configurations.annotations**
      object

      A list of annotations in the `key: "value"` format.

      Example:

      ```
      annotations:
        example: 'true'
      ```

    - **settings.configurations.excludeNames**
      array of strings

      A list of namespace patterns to exclude.

      Default: `[]`

      Example:

      ```
      excludeNames:
      - infra-test
      ```

    - **settings.configurations.includeNames**
      array of strings

      Required value

      A list of namespace patterns to include.

      Default: `[]`

      Example:

      ```
      includeNames:
      - prod-.*
      - infra-.*
      ```

    - **settings.configurations.labels**
      object

      A list of labels in the `key: "value"` format.

      Example:

      ```
      labels:
        istio-injection: 'true'
      ```

Examples

> **i** The module lifecycle stage: General Availability

Example

Add the label `extended-monitoring.deckhouse.io/enabled=true` and the annotation `foo=bar` to every namespace starting with `prod-` and `infra-`, except `infra-test`.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: namespace-configurator
spec:
  version: 1
  enabled: true
  settings:
    configurations:
    - annotations:
        foo: bar
      labels:
        extended-monitoring.deckhouse.io/enabled: "true"
      includeNames:
      - "^prod"
      - "^infra"
      excludeNames:
      - "infra-test"
```

The network-gateway module

> **i** The module lifecycle stage: General Availability

This module creates a network gateway using Kubernetes nodes. It contains a DHCP server and a SNAT manager.

Configuration

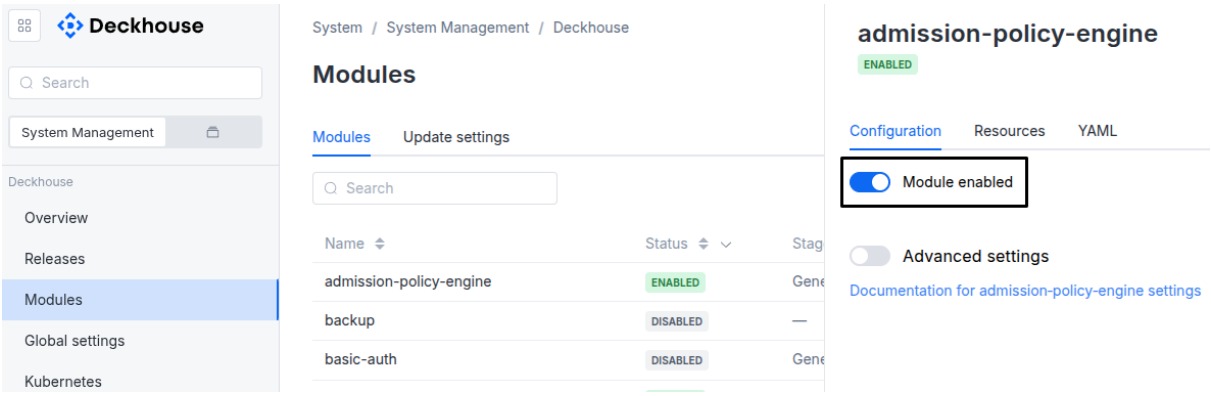
> **i** The module lifecycle stage: General Availability

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `network-gateway` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable network-gateway
  ```

- **Using ModuleConfig `network-gateway`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `network-gateway` (create it if necessary);

  Example of a manifest to enable module `network-gateway`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: network-gateway
  spec:
    enabled: true
  ```

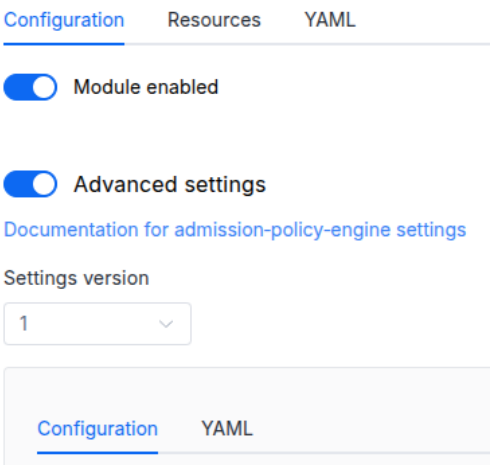
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `network-gateway` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `network-gateway` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `network-gateway`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `network-gateway` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc network-gateway
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `network-gateway` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `network-gateway`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: network-gateway
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.disableDHCP**
    boolean

    Disables the DHCP server.

    Default: `false`
  - **settings.dns**
    object

    Settings to pass to clients via DHCP.

    - **settings.dns.search**

array of strings

List of search domains.

Example:

```
search:
- office.example.com
- srv.example.com
```

- **settings.dns.servers**
  array of strings

  List of DNS servers.

  Example:

  ```
  servers:
  - 4.2.2.2
  - 8.8.8.8
  ```

- **settings.nodeSelector**
  object

  Required value

  Selects nodes that will be used to configure iptables rules and to run the DHCP server.

  The same as in the Pods' `spec.nodeSelector` parameter in Kubernetes. Instance Pods inherit this field as is.

  Example:

  ```
  nodeSelector:
    type: network-gateway
  ```

- **settings.publicAddress**
  string

  Required value

  Replaces the src of the packets outgoing from the LAN.

  Pattern: `^([0-9]{1,3}\.){3}[0-9]{1,3}$`

  Example:

  ```
  publicAddress: 10.220.203.240
  ```

- **settings.storageClass**
  string

  The name of the StorageClass to use for storing the DHCP lease.

  If omitted, the StorageClass of the existing PVC is used. If there is no PVC yet, the StorageClass will be used according to the global storageClass parameter setting.

  The global `storageClass` parameter is only considered when the module is enabled. Changing the global `storageClass` parameter while the module is enabled will not trigger disk re-provisioning.

  *Warning.** Specifying a value different from the one currently used (in the existing PVC) will result in disk re-provisioning and all data will be deleted.

  If `false` is specified, emptyDir will be forced to be used.

  Dnsmasq (underlies our DHCP server) has its own mechanisms for protecting against the duplication of IP addresses if the lease database is lost (but it is better not to lose it).

- **settings.subnet**
  string

  Required value

  The address of a local subnet that gateway serves.

  The DHCP options to pass to clients are generated based on this address:

  - Address pool — numbers starting with 50 and up to the last one.
  - Router — the subnet's first address.

  Pattern: `^([0-9]{1,3}\.){3}[0-9]{1,3}(\/([0-9]|[1-2][0-9]|3[0-2]))$`

  Example:

  ```
  subnet: 192.168.42.0/24
  ```

- **settings.tolerations**
  array of objects

Tolerations for DHCP Pods and iptables managers.

The same as in the Pods' `spec.tolerations` parameter in Kubernetes. The instance's Pods inherit this field as is.

- **settings.tolerations.effect**
  string
- **settings.tolerations.key**
  string
- **settings.tolerations.operator**
  string
- **settings.tolerations.tolerationSeconds**
  integer
- **settings.tolerations.value**
  string

The network-policy-engine module

> **i** The module lifecycle stage: General Availability

> ⚠ Do not use the module if the cilium module is enabled because it already has network policy management.

This module manages network policies.

Deckhouse implements a conservative approach to organizing the network based on elementary network backends, such as *"pure"* CNI or flannel in the `host-gw` mode. This approach is simple and reliable, and has proven itself to be the best.

The `NetworkPolicy` implementation in Deckhouse is also simple and reliable. It is based on `kube-router` in the *Network Policy Controller* mode (`--run-firewall`). In this case, `kube-router` transforms `NetworkPolicy` network policies into `iptables` rules that work in any installations (regardless of the cloud or the CNI used).

The `network-policy-engine` module deploys a `d8-system` DaemonSet in the namespace with kube-router in the Network Policy Controller mode. As a result, the Kubernetes cluster fully supports Network Policies.

The following policy description formats are supported:

- *networking.k8s.io/NetworkPolicy API*
- *network policy V1/GA semantics*
- *network policy beta semantics*

Example recipes are available here.

Third-party components

List of third-party software used in the `network-policy-engine` module:

- Kube-router 2.0.1

  License: Apache License 2.0

  A turnkey solution for Kubernetes networking with aim to provide operational simplicity and high performance.

Configuration

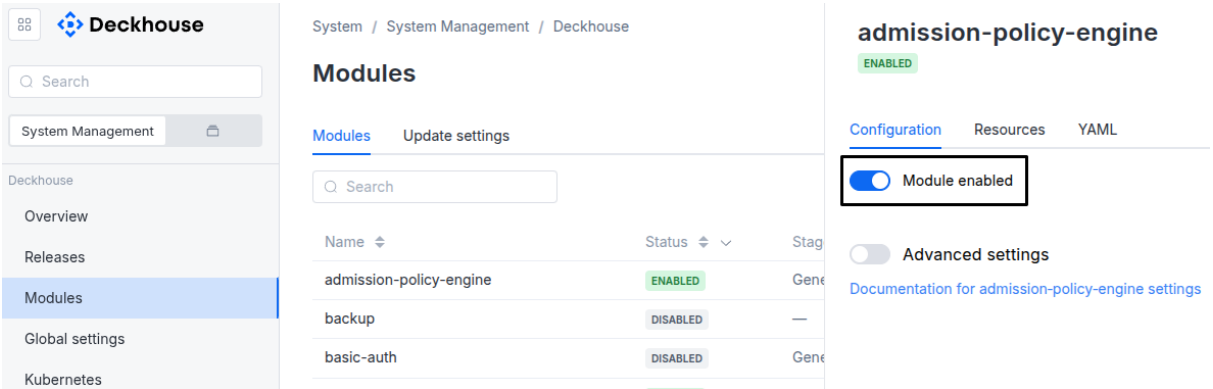
> **i** The module lifecycle stage: General Availability

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `network-policy-engine` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

Example of enabling the module:

```
d8 system module enable network-policy-engine
```

- **Using ModuleConfig `network-policy-engine`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `network-policy-engine` (create it if necessary);

  Example of a manifest to enable module `network-policy-engine`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: network-policy-engine
spec:
  enabled: true
```

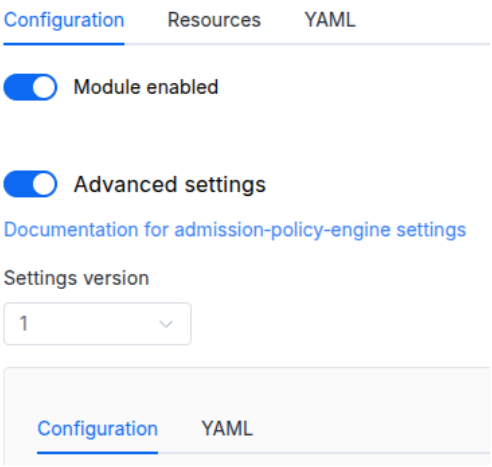
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `network-policy-engine` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `network-policy-engine` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `network-policy-engine`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `network-policy-engine` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc network-policy-engine
```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `network-policy-engine` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

  Example of a manifest for ModuleConfig `network-policy-engine`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: network-policy-engine
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Examples

> **i** The module lifecycle stage: General Availability

Examples of network policies

Deny external traffic to Pods in the namespace, but allow traffic to external resources from within the namespace

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: delete
spec:
  podSelector: {}
  egress:
  - {}
  ingress:
  - from:
    - podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

More examples are available in this repository.

The node-local-dns module

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

Module Description

This module deploys a caching DNS server on each cluster node and exports data to `Prometheus` for easy analysis of the cluster's DNS performance using the Grafana board.

The module runs the standard `CoreDNS` server (as part of a DaemonSet) on all nodes supplemented by the algorithm for configuring network and iptables rules.

Purpose

The regular DNS usage in Kubernetes poses some problems that may lead to unnecessary degradation of the service's performance:

- No request caching (in Linux, there is no caching right out-of-the-box; thus, Pods also lack it out-of-the-box).
- All the container-originated DNS requests result in a network request to the cluster DNS. Thus, even requests to resources within the same node result in network requests.
- The Pod request is first resolved in the cluster DNS zones and only then channeled to external DNS servers. For example, the request to `ya.ru` is first resolved in cluster zones such as `cluster.local`, `svc.cluster.local`, `<namespace>.svc.cluster.local`, receives negative responses, and gets appropriately resolved only on the "second" attempt.

Thus, even slight network delays may result in significant service performance degradation due to above problems.

One solution is to deploy a DNS server on each node (which is what this module does).

Outside requests (non-cached ones) are also first resolved in internal zones. Under high load (when a large number of requests are made to the same records simultaneously, as is often the case), such behavior can significantly improve the performance of DNS resolving.

Grafana dashboard

The `Kubernetes / DNS (node local)` dashboard displays:

- General graphs that allow you to evaluate the overall DNS performance.

- Graphs grouped by the node that let you analyze node data in more details if the general charts suggest that this node is unhealthy.
- Upstream-related graphs that allow you to evaluate the performance of the cluster DNS as well as the node servers specified in `/etc/resolv.conf`.

How does it work?

The module performs the following settings on each node:

- Configures an interface with the ClusterIP IP address of the `kube-dns` service.
- Starts a caching CoreDNS server that listens on this address.
- Adds a sophisticated iptables rule that uses a socket if it is open and ordinary Kubernetes magic for clusterIP if it is not.

And here is the rule itself that allows for an easy fallback:

```
-A PREROUTING -d <kube-dns IP address> -m socket --nowildcard -j NOTRACK
```

Aspects of CoreDNS configuration

Main features of the CoreDNS config:

- caching of all requests;
- forwarding all DNS requests to ClusterIP of the cluster DNS.

Configuration

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

> **⚠**
> - The module works with all `CNIs`, but in order to work correctly with `cni-cilium`, a number of conditions must be met.
> - By default, when used together with the `cni-simple-bridge` or `cni-flannel` modules, the `node-local-dns` module **does not work** for requests from `hostNetwork`. In this case, all requests go to the `kube-dns` module. You can specify the address `169.254.20.10` in the pod configuration, but if `node-local-dns` module crashes, there will be no *fallback* to the `kube-dns` module.

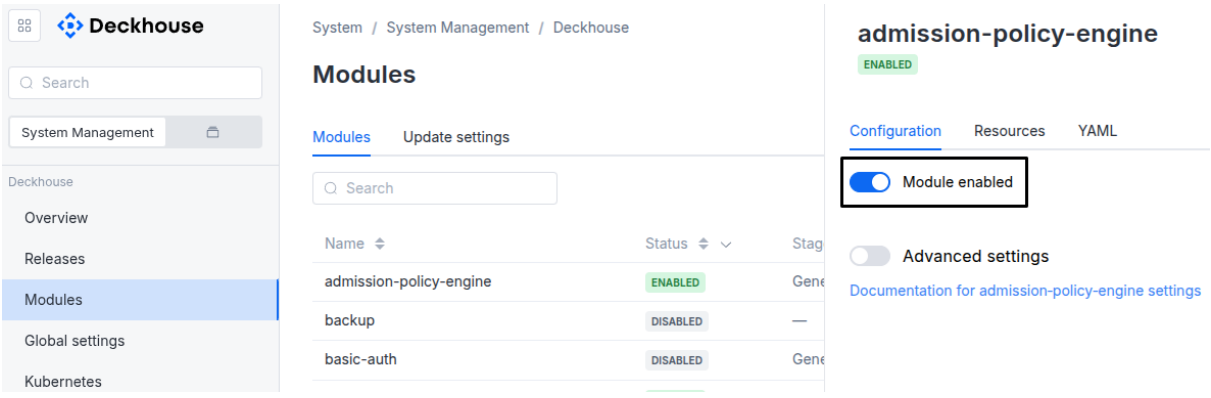
The module has 5 alerts.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `node-local-dns` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:

- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable node-local-dns
  ```

- **Using ModuleConfig `node-local-dns`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `node-local-dns` (create it if necessary);

  Example of a manifest to enable module `node-local-dns`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-local-dns
spec:
  enabled: true
```

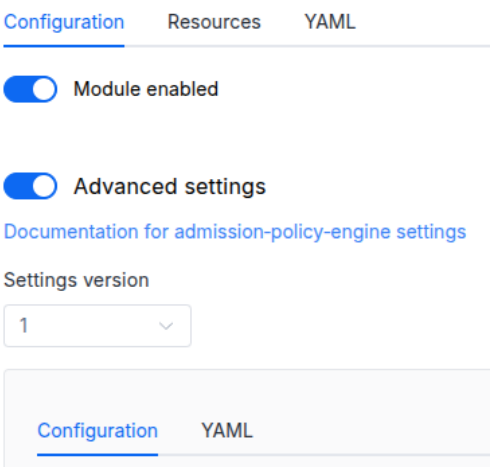
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `node-local-dns` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `node-local-dns` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `node-local-dns`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `node-local-dns` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc node-local-dns
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `node-local-dns` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `node-local-dns`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: node-local-dns
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the versions of other modules:

- `kube-dns`: any version.

Parameters

Schema version: 1

- **settings**
  object
  - **settings.disableIPv6**
    boolean

    Disables DNS resolution over IPv6 by returning empty responses for `AAAA` queries and `NXDOMAIN` for reverse IPv6 lookup requests in the `ip6.arpa` zone.

    Default: `false`
  - **settings.enableLogs**
    boolean

    Enables query logging for node-local-dns instances.

    Default: `false`
  - **settings.highAvailability**
    boolean

    Manual failover mode management. Within this module, the option affects the DNS records caching mechanism on nodes:

    - `true`: upon the expiration of the TTL for an individual DNS record, node-local-dns will continue to return the “stale” value immediately while simultaneously requesting a fresh value from the upstream kube-dns. Thus, in the event of issues with the nodes where kube-dns is running, applications will continue to receive responses to DNS queries with expired TTL, but without delay.
    - `false`: upon the expiration of the TTL for an individual DNS record, node-local-dns will continue to return the “stale” value, but only after querying the upstream kube-dns. Thus, in the event of issues with the nodes where kube-dns is running, applications will continue to receive DNS responses with expired TTL, but with potentially significant delay. This approach is suitable in highly dynamic environments where Services and Namespaces are frequently recreated.

    By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

    Example:

    ```
    highAvailability: true
    ```

Examples

> **i**
> The module lifecycle stage: General Availability
> The module has requirements for installation

An example of configuring a custom DNS for a Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: dns-example
spec:
  dnsPolicy: "None"
  dnsConfig:
    nameservers:
      - 169.254.20.10
  containers:
    - name: test
      image: nginx
```

Here you can learn more about DNS configuring.

Configuration example

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-local-dns
spec:
  version: 1
  enabled: true
  settings:
    enableLogs: true
```

Managing nodes

> **i** The module lifecycle stage: General Availability

Primary functions

The `node-manager` module is responsible for managing nodes and has the following primary functions:

1. Managing multiple nodes as a related group (**NodeGroup**):
   - The ability to define metadata that are inherited by all nodes in the group.
   - Monitoring of a group as a single entity (grouping nodes on graphs by groups, grouping alerts about node unavailability, alerts about the unavailability of N or N% of nodes in a group).
2. **Chaos monkey** — the systemic termination of nodes. This feature tests the resilience of cluster elements and running applications.
3. Delaying the reboot or shutdown of a node while critical pods are running on it.
4. Installing, updating, and configuring the node software (containerd, kubelet, etc.), connecting the node to the cluster:
   - Installing operating system (see the list of supported OS) regardless of the infrastructure used (any cloud/any hardware).
   - The operating system's basic setup (disabling auto-update, installing the necessary packages, configuring logging parameters, etc.).
   - Configuring kubernetes-api-proxy (and the system for automatically updating the list of upstreams) to balance node (kubelet) requests over API servers.
   - Installing and configuring CRI containerd and Kubernetes, adding the node to the cluster.
   - Managing node updates and their downtime (disruptions):
     - Automatic determination of a valid minor Kubernetes version for a node group based on its settings (the kubernetesVersion parameter specified for a group), the default version for the whole cluster, and the current control-plane version (no nodes can be updated ahead of the control-plane update).
     - Only one node of a group can be updated at a time, and only if all the nodes in the group are available.
     - There are two types of node updates:
       - regular updates – always performed automatically;
       - disruption-involving updates (such as updating the kernel, switching docker versions, major change of the kubelet versions, etc.) – you can choose manual or automatic mode. If automatic disruptive updates are enabled, the node is drained before the update (this functionality can be disabled).
   - Monitoring the status and progress of the update.
5. Cluster scaling.

   - Autoscaling.

     Available with supported cloud providers (learn more); not available for static nodes. Cloud providers support automatic creation or deletion of virtual machines, joining them to or disjoining them from the cluster.

   - Maintaining the desired number of nodes in a group.

     Available for both cloud providers and static nodes (when using Cluster API Provider Static).

6. Managing Linux users on nodes.
7. Managing GPU resources on nodes:
   - Automatic detection and enablement of NVIDIA GPU.
   - Configurable GPU sharing modes per NodeGroup: Exclusive, TimeSlicing, MIG.
   - Monitoring integration — ready-made Grafana dashboards are available to visualize key GPU metrics.

Nodes are managed through the NodeGroup resource, and each node group performs specific tasks. Below are examples of node groups pooled by their tasks performed:

- a group of master nodes;
- a group of nodes for routing HTTP(s) traffic (front nodes);
- a group of monitoring-related nodes;
- a group of application nodes (the so-called worker nodes), etc.

The nodes belonging to the group have common parameters and are configured automatically according to the group's parameters. Deckhouse scales groups by adding, excluding, and updating their nodes. Both Static nodes deployed in the cloud and static (bare-metal or virtual machine) nodes can be combined into a single group. It paves the way for the hybrid clusters, in which you can scale nodes on physical servers via cloud nodes.

Operations on the cloud infrastructure are performed by means provided by supported cloud providers. If there is no support for the desired cloud platform, you can use its resources as static nodes.

The static nodes (e.g., bare metal servers) are handled by the Cluster API Provider Static (CAPS).

The following Managed Kubernetes services are supported (note that some service functionality may be unavailable):

- Google Kubernetes Engine (GKE);
- Elastic Kubernetes Service (EKS).

Node types

The following node types that can be worked with within a node group (resource NodeGroup) are supported:

- `CloudEphemeral` — the nodes are automatically ordered, created, and deleted in the configured cloud provider.
- `CloudPermanent` — the nodes differ in that their configuration is not taken from the custom resource nodeGroup, but from a special resource `<PROVIDER>ClusterConfiguration` (for example, AWSClusterConfiguration for AWS). Also, an important difference is that to apply node configuration, you need to run `dhctl converge` (by running Deckhouse installer). An example of a CloudPermanent node of a cloud cluster is a cluster master node.
- `CloudStatic` — a static node (created manually) hosted in the cloud integrated with one of the cloud providers. This node has the CSI running, and it is managed by the cloud-controller-manager. The `Node` object automatically gets the information about the cloud zone and region. Also, if a node gets deleted from the cloud, its corresponding Node object will be deleted in a cluster.
- `Static` — a static node hosted on a bare metal or virtual machine. In the case of a cloud environment, the `cloud-`

`controller-manager` does not manage the node even if one of the cloud providers is enabled. Learn more about working with static nodes...

Node grouping and group management

Grouping and managing nodes as a related group mean that all nodes in the group will have the same metadata derived from the `NodeGroup` custom resource.

The following monitoring patterns are available for node groups:

- with grouping of node parameters on the graphs for the group;
- with grouping of alerts about node unavailability;
- with alerts about unavailability of N nodes or N% of nodes in the group, etc.

Delaying node reboot or shutdown while critical pods are running

The `d8-shutdown-inhibitor` service in `node-manager` delays node shutdown or reboot if stateful applications or virtual machines (VMs) are running on the node. The delay is triggered if pods with the label `pod.deckhouse.io/inhibit-node-shutdown` are running on the node. DKP delays the shutdown or reboot of the node so that the user can migrate critical pods or terminate them in a controlled manner. This is useful, for example, for stateful workloads and other scenarios where data loss must be avoided.

How it works and key features

The `d8-shutdown-inhibitor` service prevents a node from shutting down as long as there are pods on it with the `Running` status that are labeled `pod.deckhouse.io/inhibit-node-shutdown`. The `GracefulShutdownPostpone` condition is used for this blocking. Additionally, the node is cordoned off during the blocking period to prevent the scheduler from scheduling new pods on it. The kubelet reads the node's `GracefulShutdownPostpone` condition and does not begin evicting pods marked with this label as long as the condition's `status` is `True` and `reason` is `PodsWithLabelAreRunningOnNode`. If there are pods on the node with the label `pod.deckhouse.io/inhibit-node-shutdown`, a corresponding message is displayed in the node console (wall).

Delaying mechanism features in DKP

The delay in restarting or shutting down a node while critical pods are running on it has the following features in DKP:

- A label must be explicitly set on each pod that is to block shutdown. If pods are created via a Deployment, the label is specified in the pod template.
- This mechanism works only with a modified kubelet version included in DKP. The regular Kubernetes does not support this behavior.
- The maximum block duration (`InhibitDelayMaxSec`) is `3` days. After this period expires, the shutdown may proceed regardless of whether there are pods on the node with the label `pod.deckhouse.io/inhibit-node-shutdown`.
- This mechanism does not cancel the scheduling of disruptive updates in `node-manager`. It only delays the actual reboot of the node at the time of shutdown.

> ⚠ To decide whether to block a node shutdown, DKP additionally queries the NodeGroup. If the current node belongs to the `master` group and it is the only master node in the cluster, the shutdown block will not be applied to it.

For information on enabling the delay mechanism, refer to the section "How to delay a node reboot while critical pods are running on it".

Automatic deploying, configuring and updating Kubernetes nodes

Automatic deployment (partially supported for *static/hybrid* nodes/clusters), configuration, and continuing software updates are supported for all cluster types (cloud or bare metal based).

Deploying Kubernetes nodes

Deckhouse automatically deploys cluster nodes by performing the following **idempotent** operations:

- Configuring the OS and optimizing it for containerd and Kubernetes:
  - Installing the needed packages from the distribution's repository.
  - Configuring kernel parameters, logging, log rotation, and other system parameters.
- Installing the appropriate versions of containerd and kubelet; adding the node to the Kubernetes cluster.
- Configuring kubernetes-api-proxy and updating the list of upstream resources for balancing node requests to the Kubernetes API.

Keeping nodes up-to-date

Two types of updates can be applied to keep cluster nodes up to date:

- **Regular**. These updates are always applied automatically and do not cause the node to stop or reboot.
- **Disruptive**. An example of such updates is a kernel or containerd version update, a major kubelet version change, etc. For this type of updates you can choose manual or automatic mode (disruptions parameter section). In automatic mode, the node is gracefully drained before the update.

Only one node of the group is updated at a time, and only if all the nodes in the group are available.

The `node-manager` module has a set of built-in metrics for monitoring the update process, alerting about issues with the update, or when a decision to proceed needs to be made by an administrator.

Working with nodes on supported cloud platforms

Each supported cloud provider can provision nodes in an automated manner. For this, you have to specify the required parameters for each node or a group of nodes.

Depending on the provider, these parameters can include:

- type of a node, the number of processor cores, the amount of RAM;
- disk size;
- security settings;
- connected networks;
- other parameters.

Creating, starting, and connecting virtual machines to the cluster are performed automatically.

Scaling nodes in the cloud

Two methods of scaling nodes in a group are supported:

- **Automatic scaling**

  If there is a resource shortage or pods in the `Pending` state are present, nodes will be added to the group. If there is no load on one or more nodes, they will be removed from the cluster. Autoscaling takes into account the group priority (the group with the highest priority will be scaled first).

  To enable automatic node scaling, specify *different* non-zero values for the minimum and maximum number of nodes in the group.

- **Fixed number of nodes**

  In this case, Deckhouse will maintain the specified number of nodes (e.g., by provisioning new nodes when nodes fail).

  To set a fixed number of nodes in a group and disable automatic scaling, specify the *same* values for minPerZone and maxPerZone.

Working with static nodes

When working with static nodes, some features of the `node-manager` module are limited:

- **No node provisioning.** Resources (bare metal servers, virtual machines, linked resources) are provisioned manually. Subsequent configuration of resources (connecting a node to the cluster, setting up monitoring, etc.) is performed either fully (similar to nodes in the cloud) or partly automatically.
- **No node autoscaling.** Maintaining the specified number of nodes in a group is supported when using Cluster API Provider Static (parameter staticInstances.count). That is, Deckhouse will attempt to maintain the specified number of nodes in the group, cleaning up unnecessary nodes and setting up new ones as needed (by picking them from StaticInstance resources that are in *Pending* state).

Configuring/clearing up a node, joining it to a cluster, and disjoining it can be done in the following ways:

- **Manually** using the pre-made scripts.

  To configure the server (VM) and join a node to the cluster, a special bootstrap script must be downloaded and run. This script is generated for each group of static nodes (each `NodeGroup` resource). It is located in the `d8-cloud-instance-manager/manual-bootstrap-for-<NAME-NODEGROUP>` secret. See an example of how to add a static node to a cluster.

  To disjoin a cluster node and clean up the server (virtual machine), you need to run the `/var/lib/bashible/cleanup_static_node.sh` script (it is present on each static node). An example of decommissioning a cluster node and cleaning up the server can be found in FAQ.

- **Automatically** using Cluster API Provider Static.

  Cluster API Provider Static (CAPS) connects to the server (VM) using StaticInstance and SSHCredentials resources, configures, and joins the node into the cluster.

  If necessary (for example, if the StaticInstance resource associated with the server is deleted or the number of group nodes is reduced), the Cluster API Provider Static connects to the cluster node, clears it, and disconnects it from the cluster.

- **Manually with subsequent transfer of the Node under automatic control** of Cluster API Provider Static.

  > **i** This feature is available starting with Deckhouse 1.63.

  To transfer an existing cluster node under CAPS management, it is necessary to prepare the StaticInstance and SSHCredentials resources for this node, as with automatic management in the point above, however the StaticInstance resource must additionally be annotated as `static.node.deckhouse.io/skip-bootstrap-phase: ""`.

Cluster API Provider Static

Cluster API Provider Static (CAPS) is an implementation of a declarative management provider for static nodes (bare metal servers or virtual machines) for the Kubernetes Cluster API. Essentially, CAPS is an additional layer of abstraction to the existing Deckhouse functionality that provides automatic static node configuration and cleanup using scripts generated for each node group (see Working with Static Nodes).

CAPS features are as follows:

- configuring a bare metal server (or virtual machine) to join a Kubernetes cluster;
- joining a node to the Kubernetes cluster;
- disjoining the node from the Kubernetes cluster;
- cleaning the bare metal server (or virtual machine) after disjoining the node from the Kubernetes cluster.

CAPS uses the following CustomResource when operating:

- **StaticInstance.** Each `StaticInstance` resource details a specific host (server, VM) that is managed using CAPS.
- **SSHCredentials**. Contains the SSH credentials required to connect to the host (`SSHCredentials` data is specified in the credentialsRef parameter of the `StaticInstance` resource).
- **NodeGroup**. The staticInstances parameter section defines the required number of nodes in the group and the filter for the `StaticInstance` resources that can be used in the group.

CAPS is enabled automatically if the NodeGroup has the staticInstances parameter section filled in. If the `staticInstances` parameter section in the `NodeGroup` is empty, then setting up and cleaning up nodes for that group is done *manually* rather than using CAPS (see adding a static node to a cluster and cleaning up a node).

The workflow for dealing with static nodes when using Cluster API Provider Static (CAPS) (see the adding a node example) is as follows:

1. **Preparing resources.**

   Before bringing a bare metal server or virtual machine under CAPS control, some preliminary steps may be necessary:

   - Preparing the storage system, adding mount points, and so on.
   - Installing the OS-specific packages.
   - Configuring the network. For example, configuring the network between the server and the cluster nodes.
   - Configuring SSH access to the server, creating a user that is a member of sudoers group. A good practice is to create a separate user and unique keys for each server.

2. **Creating a SSHCredentials resource.**

   The `SSHCredentials` resource contains the parameters required by CAPS to connect to a server via SSH. A single `SSHCredentials` resource can be used to connect to multiple servers, but it is a good practice to create unique users and access keys to connect to each server. In this case, the `SSHCredentials` resource will also be unique for each server.

3. **Creating a StaticInstance resource.**

   For every server (VM) in the cluster, an individual `StaticInstance` resource is created. It contains the IP address for connecting and a link to the `SSHCredentials` resource with the data to be used for connecting.

   The following is a list of possible `StaticInstance` states and its associated servers (VMs) and cluster nodes:

   - `Pending`. The server is not configured and there is no associated node in the cluster.
   - `Bootstrapping`. The procedure for configuring the server (VM) and connecting the node to the cluster is in progress.
   - `Running`. The server is configured and the associated node is added to the cluster.
   - `Cleaning`. The procedure of cleaning up the server and disconnecting the node from the cluster is in progress.

   > **i** You can transfer the existing manually-bootstrapped cluster node under CAPS management by annotating its StaticInstance with `static.node.deckhouse.io/skip-bootstrap-phase: ""`.

4. **Creating a NodeGroup resource.**

   When using CAPS, you have to focus on the nodeType parameter (must be `Static`) of the `NodeGroup` resource as well as the staticInstances parameter section.

   The staticInstances.labelSelector parameter section defines a filter that CAPS applies to select the `StaticInstance` resources to be used for a group. The filter allows only certain `StaticInstance` to be used for specific node groups, and also allows a single `StaticInstance` to be used in different node groups. You can choose not to define a filter to use any available `StaticInstance` for a node group.

   The staticInstances.count parameter specifies the desired number of nodes in the group. When the parameter changes, CAPS starts adding or removing the desired number of nodes (this process runs in parallel).

Using the data in the staticInstances parameter section, CAPS attempts to maintain the specified number of nodes in the group (count parameter). If a node needs to be added to the group, CAPS selects the resource StaticInstance that matches the filter and is in the `Pending` state, configures the server (VM), and joins the node to the cluster. If a node needs to be removed from the group, CAPS selects the StaticInstance that is in the `Running` state, cleans up the server (VM) and disconnects the node from the cluster (the corresponding `StaticInstance` then goes to the `Pending` state and can be reused).

Custom node settings

The NodeGroupConfiguration resource allows you to automate actions on group nodes. It supports running bash scripts on nodes (you can use the bashbooster command set) as well as the Go Template templating engine, and is a great way to automate operations such as:

- Installing and configuring additional OS packages.

  Examples:

  - installing the plugin for kubectl;
  - installing containerd with Nvidia GPU support.

- Updating the OS kernel to a specific version.

  Examples:

  - Debian kernel update;
  - CentOS kernel update.

- Modifying OS parameters.

Examples:

  - customizing the sysctl parameter;
  - adding a root certificate.

- Collecting information on a node and carrying out other similar tasks.

The `NodeGroupConfiguration` resource allows you to assign priority to scripts being run or limit them to running on specific node groups and OS types.

The script code is stored in the content of the resource. When a script is created on a node, the contents of the `content` parameter are fed into the Go Template templating engine. The latter embeds an extra layer of logic when generating a script. When parsed by the templating engine, a context with a set of dynamic variables becomes available.

The following variables are supported by the templating engine:

- `.cloudProvider` (for node groups of nodeType `CloudEphemeral` or `CloudPermanent`) — cloud provider dataset.

  Example of data...

  ```
  cloudProvider:
    instanceClassKind: OpenStackInstanceClass
    machineClassKind: OpenStackMachineClass
    openstack:
      connection:
        authURL: https://cloud.provider.com/v3/
        domainName: Default
        password: p@ssw0rd
        region: region2
        tenantName: mytenantname
        username: mytenantusername
      externalNetworkNames:
      - public
      instances:
        imageName: ubuntu-24-04-cloud-amd64
        mainNetwork: kube
        securityGroups:
        - kube
        sshKeyPairName: kube
      internalNetworkNames:
      - kube
      podNetworkMode: DirectRoutingWithPortSecurityEnabled
    region: region2
    type: openstack
    zones:
    - nova
  ```

- `.cri` — the CRI in use (starting with Deckhouse 1.49, only `Containerd` is supported).
- `.kubernetesVersion` — the Kubernetes version in use.
- `.nodeUsers` — the dataset with information about node users added via the NodeUser.

  Example of data...

  ```
  nodeUsers:
  - name: user1
    spec:
      isSudoer: true
      nodeGroups:
      - '*'
      passwordHash: PASSWORD_HASH
      sshPublicKey: SSH_PUBLIC_KEY
      uid: 1050
  ```

- `.nodeGroup` — node group dataset.

  Example of data...

```
nodeGroup:
  cri:
    type: Containerd
  disruptions:
    approvalMode: Automatic
  kubelet:
    containerLogMaxFiles: 4
    containerLogMaxSize: 50Mi
    resourceReservation:
      mode: "Off"
  kubernetesVersion: "1.31"
  manualRolloutID: ""
  name: master
  nodeTemplate:
    labels:
      node-role.kubernetes.io/control-plane: ""
      node-role.kubernetes.io/master: ""
    taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
  nodeType: CloudPermanent
  updateEpoch: "1699879470"
```

An example of using variables in a template:

```
{{- range .nodeUsers }}
echo 'Tuning environment for user {{ .name }}'
# Some code for tuning user environment
{{- end }}
```

An example of using bashbooster commands:

```
bb-event-on 'bb-package-installed' 'post-install'
post-install() {
  bb-log-info "Setting reboot flag due to kernel was updated"
  bb-flag-set reboot
}
```

The progress of script execution can be seen on the node in the bashible service log using the command:

```
journalctl -u bashible.service
```

The scripts themselves are located on the node in the directory `/var/lib/bashible/bundle_steps/`.

The service decides to re-run the scripts by comparing the single checksum of all files located at `/var/lib/bashible/configuration_checksum` with the checksum located in the `kubernetes` cluster in the `configuration-checksums` secret namespace `d8-cloud-instance-manager`.

You can see the checksum using the following command:

```
d8 k -n d8-cloud-instance-manager get secret configuration-checksums -o yaml
```

The service compares checksums every minute.

The checksum in the cluster changes every 4 hours, thereby re-running the scripts on all nodes. Forcing the execution of bashible on a node can be done by deleting the file with the script checksum using the following command:

```
rm /var/lib/bashible/configuration_checksum
```

Features of writing scripts

When writing your own scripts, it is important to consider the following features of their use in Deckhouse:

1. Scripts in deckhouse are executed once every 4 hours or more often based on external triggers. Therefore, it is important to write scripts in such a way that they check the need for their changes in the system before performing actions, thereby not making changes every time they are launched.
2. There are built-in scripts that perform various actions, including installing and configuring services. This is important to consider when choosing the priority of custom scripts. For example, if a script is planned to restart a service, then this script should be called after the service installation script, since otherwise it will not be able to execute when deploying a new node.

Useful features of some scripts:

- `032_configure_containerd.sh`: Merges the containerd configuration files and **restarts** the service. The location of the configuration files depends on the CRI used on the cluster nodes:
  - `/etc/containerd/conf.d/*.toml`: For containerd v1.

- `/etc/containerd/conf2.d/*.toml`: For containerd v2.

Note that directories containing configuration files are not created automatically. Files must be added to these directories in scripts with a priority of less than `32`.

Chaos Monkey

The instrument (you can enable it for each `NodeGroup` individually) for unexpected and random termination of nodes in a systemic manner. Chaos Monkey tests the resilience of cluster elements, applications, and infrastructure components.

Monitoring

DKP exports the group availability Prometheus metrics for each `NodeGroup`.

What information does Prometheus collect and in what form?

Metrics have the `d8_node_group_` prefix.

All the collected metrics have a label that allow you to identify the node group: `node_group_name`.

The following metrics are collected for each node group:

- `d8 node group ready` — the number of Kubernetes nodes in the group that are `Ready`.
- `d8 node group nodes` — the number of Kubernetes nodes (in any state) in the group.
- `d8 node group instances` — the number of instances (in any state) in the group.
- `d8 node group desired` — the number of desired `Machines` in the group.
- `d8 node group min` — the minimum number of instances in the group.
- `d8 node group max` — the maximum number of instances in the group.
- `d8 node group up to date` — the number of up-to-date nodes in the group.
- `d8 node group standby` — the number of over provisioned instances (see the standby parameter) in the group.
- `d8_node_group_has_errors` — equals to 1 if there are errors in the group.

Third-party components

List of third-party software used in the `node-manager` module:

- Gardener Machine Controller Manager v0.36.0

  License: Apache License 2.0

  An operator that manages VMs as a kubernetes custom resource.

- Cluster API 1.11.3

  License: Apache License 2.0

  Kubernetes Cluster API for declarative cluster lifecycle management.

- Cluster Autoscaler

  License: Apache License 2.0

  Kubernetes autoscaler that automatically adjusts the size of the cluster.

Configuration

> **i** The module lifecycle stage: General Availability

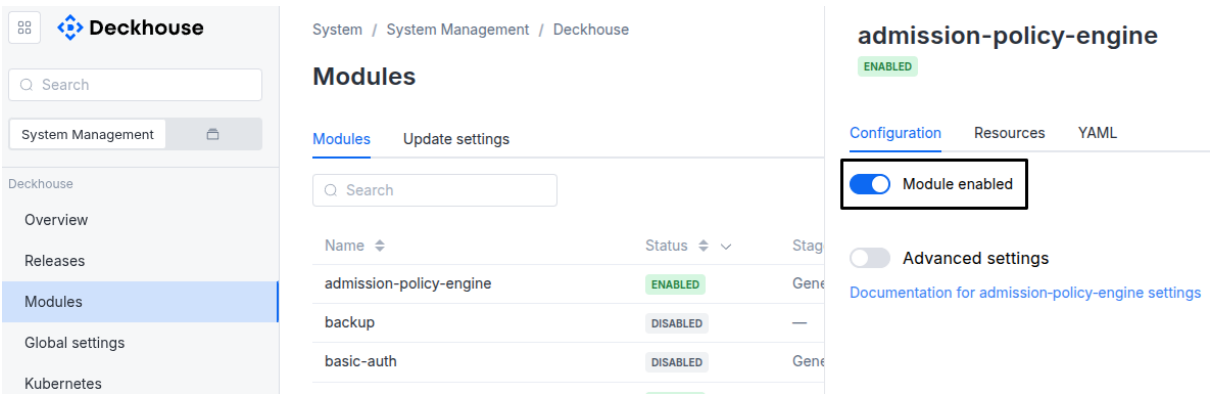
The module has 37 alerts.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `node-manager` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable node-manager
  ```

- **Using ModuleConfig `node-manager`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `node-manager` (create it if necessary);

  Example of a manifest to enable module `node-manager`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: node-manager
  spec:
    enabled: true
  ```

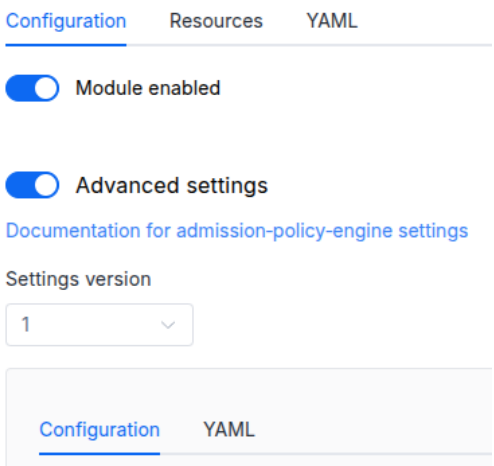
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `node-manager` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `node-manager` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `node-manager`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `node-manager` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc node-manager
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `node-manager` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `node-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: node-manager
spec:
  version: 2
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Delete `allowedKubernetesVersions` and `allowedBundles`.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.capiEmergencyBrake**
    boolean

    Set to ‘true’ to disable CAPI in case of potentially cluster-damaging changes.

    > **Warning.** Use this flag only if you know why you are doing it and clearly understand the consequences.

    Default: `false`

    Example:

    ```
    capiEmergencyBrake: false
    ```
  - **settings.earlyOomEnabled**
    boolean

    Set to ‘false’ to disable early OOM killer in case it behaves incorrectly.

    Default: `true`

    Example:

    ```
    earlyOomEnabled: false
    ```
  - **settings.instancePrefix**
    string

    The prefix to use when creating instances via the corresponding cloud provider module.

    The default value can be calculated based on the `ClusterConfiguration` struct if the cluster is installed via the Deckhouse installer.

    Example:

    ```
    instancePrefix: kube
    ```
  - **settings.mcmEmergencyBrake**
    boolean

    Set to ‘true’ to disable machine-controller-manager in case of potentially cluster-damaging changes.

    > **Warning!** Use this flag only if you know what you are doing and clearly understand the consequences.

    Default: `false`

    Example:

    ```
    mcmEmergencyBrake: false
    ```

Custom resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

Instance

Scope: Cluster
Version: v1alpha1

Describes an implementation-independent ephemeral machine resource.

- **status**
  object
  - **status.bootstrapStatus**
    object

    Information about instance bootstrapping process.

    - **status.bootstrapStatus.description**
      string

      Description about getting logs.

    - **status.bootstrapStatus.logsEndpoint**
      string

      http endpoint for getting bootstrap logs.

  - **status.classReference**
    object

    The reference to the `InstanceClass` object. It is unique for each `cloud-provider-*` module.

    - **status.classReference.kind**
      string

      The object type (e.g., `OpenStackInstanceClass`). The object type is specified in the documentation of the corresponding `cloud-provider-` module.

      Allowed values: `OpenStackInstanceClass`, `GCPInstanceClass`, `VsphereInstanceClass`, `AWSInstanceClass`, `YandexInstanceClass`, `AzureInstanceClass`, `VCDInstanceClass`, `ZvirtInstanceClass`, `DynamixInstanceClass`, `HuaweiCloudInstanceClass`, `DVPInstanceClass`

    - **status.classReference.name**
      string

      The name of the required `InstanceClass` object (e.g., `finland-medium`).

  - **status.currentStatus**
    object

    Current status of the instance object.

    - **status.currentStatus.lastUpdateTime**
      string

      Last update time of current status.

    - **status.currentStatus.phase**
      string

      InstancePhase is a label for the condition of a instance at the current time.

  - **status.lastOperation**
    object

    Last operation refers to the status of the last operation performed.

    - **status.lastOperation.description**
      string

      Description of the current operation.

        - **status.lastOperation.lastUpdateTime**
          string

          Last update time of current operation.

        - **status.lastOperation.state**
          string

          State of operation.

        - **status.lastOperation.type**
          string

          Type of operation.

    - **status.machineRef**
      object

      Reference to instance in the cloud.

        - **status.machineRef.apiVersion**
          string

          APIVersion defines the versioned schema of this representation of an reference object.

        - **status.machineRef.kind**
          string

          Kind is a string value representing the REST resource referenced object represents.

        - **status.machineRef.name**
          string

          Name of referenced object.

        - **status.machineRef.namespace**
          string

          Namespace of referenced object.

    - **status.nodeRef**
      object

      Reference to the kubernetes node object.

        - **status.nodeRef.name**
          string

          Node name

InstanceTypesCatalog

Scope: Cluster
Version: v1alpha1

Contains information about discovered data from cloud.

- **instanceTypes**
  array of objects

  Required value

  List of available instance types.

    - **instanceTypes.cpu**
      integer or string

      Required value

      The number of CPUs for the instance type.

      Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

    - **instanceTypes.memory**
      integer or string

      Required value

      The amount of RAM a instance type has.

      Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

- **instanceTypes.name**
  string

  Instance type name.

- **instanceTypes.rootDisk**
  integer or string

  Root disk size.

  Pattern: `^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$`

NodeGroup

Short names: ng

Scope: Cluster

- v1
- v1alpha2
- v1alpha1

Describes the runtime parameters of the node group.

Example:

```
# NodeGroup for cloud nodes in AWS.
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: test
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    zones:
      - eu-west-1a
      - eu-west-1b
    minPerZone: 1
    maxPerZone: 2
    classReference:
      kind: AWSInstanceClass
      name: test
  nodeTemplate:
    labels:
      tier: test
---
# NodeGroup for static nodes on bare metal servers (or VMs).
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
```

- **metadata**
  object
  - **metadata.name**
    string

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

    Maximum length: `42`

- **status**
  object
  - **status.conditionSummary**
    object
    - **status.conditionSummary.ready**
      string

      Status of the condition summary.

Allowed values: `True`, `False`

- **status.conditionSummary.statusMessage**
  string

  Status message about group handling.

- **status.conditions**
  array of objects
  - **status.conditions.lastTransitionTime**
    string

    Last time the condition transit from one status to another.

  - **status.conditions.message**
    string

    Human readable message indicating details about last transition.

  - **status.conditions.status**
    string

    Required value

    Status of the condition, one of True, False.

  - **status.conditions.type**
    string

    Required value

    Type of node group condition.

- **status.deckhouse**
  object
  - **status.deckhouse.observed**
    object

    Contains last timestamp when the resource change was noted by the operator and its checksum.

    - **status.deckhouse.observed.checkSum**
      string

      The checksum of the last observed resource.

    - **status.deckhouse.observed.lastTimestamp**
      string

      Last timestamp when a change of the resource was noted by the operator.

  - **status.deckhouse.processed**
    object

    Contains last timestamp when the resource was applied in the cluster by the operator and its checksum.

    - **status.deckhouse.processed.checkSum**
      string

      The checksum of the last applied resource.

    - **status.deckhouse.processed.lastTimestamp**
      string

      Last timestamp when the resource was applied in the cluster.

  - **status.deckhouse.synced**
    string

    True if last observed version of the resource was successfully applied in the cluster.

- **status.desired**
  integer

  Number of desired machines in the group.

- **status.error**
  string

  Error message about possible problems with the group handling.

- **status.instances**

integer

Number of instances (in any state) in the group.

- **status.kubernetesVersion**
  string

  The current version of kubernetes on the nodes, or the version to which the nodes will be upgraded.

- **status.lastMachineFailures**
  array of objects
  - **status.lastMachineFailures.lastOperation**
    object
    - **status.lastMachineFailures.lastOperation.description**
      string

      Last operation’s description.

    - **status.lastMachineFailures.lastOperation.lastUpdateTime**
      string

      Timestamp of last status update for operation.

    - **status.lastMachineFailures.lastOperation.state**
      string

      Machine’s operation state.

    - **status.lastMachineFailures.lastOperation.type**
      string

      Type of operation.

  - **status.lastMachineFailures.name**
    string

    Machine’s name.

  - **status.lastMachineFailures.ownerRef**
    string

    Machine owner’s name.

  - **status.lastMachineFailures.providerID**
    string

    Machine’s ProviderID.

- **status.max**
  integer

  Maximum amount of instances in the group.

- **status.min**
  integer

  Minimal amount of instances in the group.

- **status.nodes**
  integer

  Number of Kubernetes nodes (in any state) in the group.

- **status.ready**
  integer

  Number of ready Kubernetes nodes in the group.

- **status.standby**
  integer

  Number of overprovisioned instances in the group.

- **status.upToDate**
  integer

  Number of up-to-date nodes in the group.

- **spec**
  object

  Required value

- **spec.chaos**
  object

  Chaos monkey settings.

  Example:

  ```
  chaos:
    mode: DrainAndDelete
    period: 24h
  ```

  - **spec.chaos.mode**
    string

    The chaos monkey mode:

    - `DrainAndDelete` — drains and deletes a node when triggered;
    - `Disabled` — leaves this NodeGroup intact.

    Default: `Disabled`

    Allowed values: `Disabled`, `DrainAndDelete`

  - **spec.chaos.period**
    string

    The time interval to use for the chaos monkey.

    It is specified as a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

    Default: `6h`

    Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

- **spec.cloudInstances**
  object

  Parameter for provisioning the cloud-based VMs.

  > **i** **Caution!** Can only be used together with `nodeType: CloudEphemeral`.

  - **spec.cloudInstances.classReference**
    object

    Required value

    The reference to the `InstanceClass` object. It is unique for each `cloud-provider-*` module.

    - **spec.cloudInstances.classReference.kind**
      string

      The object type (e.g., `OpenStackInstanceClass`). The object type is specified in the documentation of the corresponding `cloud-provider-` module.

      Allowed values: `OpenStackInstanceClass`, `GCPInstanceClass`, `VsphereInstanceClass`, `AWSInstanceClass`, `YandexInstanceClass`, `AzureInstanceClass`, `VCDInstanceClass`, `ZvirtInstanceClass`, `DynamixInstanceClass`, `HuaweiCloudInstanceClass`, `DVPInstanceClass`

    - **spec.cloudInstances.classReference.name**
      string

      The name of the required `InstanceClass` object (e.g., `finland-medium`).

  - **spec.cloudInstances.maxPerZone**
    integer

    Required value

    The maximum number of instances for the group in each zone.

    This value is used as the upper bound in cluster-autoscaler.

    Allowed values: `0 <= X`

  - **spec.cloudInstances.maxSurgePerZone**
    integer

The maximum number of instances to rollout simultaneously in the group in each zone.

Default: `1`

Allowed values: `0 <= X`

- **spec.cloudInstances.maxUnavailablePerZone**
  integer

  The maximum number of unavailable instances (during rollout) in the group in each zone.

  Default: `0`

  Allowed values: `0 <= X`

- **spec.cloudInstances.minPerZone**
  integer

  Required value

  The minimum number of instances for the group in each zone.

  This value is used in the `MachineDeployment` object and as a lower bound in cluster-autoscaler.

  Allowed values: `0 <= X`

- **spec.cloudInstances.priority**
  integer

  Priority of the node group.

  When scaling a cluster, the autoscaler will first select node groups with a higher priority set. If several node groups have the same priority, the autoscaler randomly selects a group of them. You can control the rate at which nodes are switched to lower-priority groups on demand (for more details, see the FAQ).

  Using priorities can be convenient to prefer ordering cheaper nodes (for example, spot instances) over more expensive ones.

- **spec.cloudInstances.quickShutdown**
  Deprecated
  boolean

  Lowers CloudEphemeral machine drain timeout to 5 minutes.

- **spec.cloudInstances.standby**
  integer or string

  The summary number of overprovisioned nodes for this `NodeGroup` in all zones.

  An overprovisioned node is a cluster node on which resources are reserved that are available at any time for scaling. The presence of such a node allows the cluster autoscaler not to wait for node initialization (which may take several minutes), but to immediately place a load on it.

  The value can be an absolute number (for example, 2) or a percentage of desired nodes (for example, 10%). If a percentage is specified, the absolute number is calculated based on the percentage of the maximum number of nodes (the maxPerZone parameter) rounded down, but not less than one.

  Pattern: `^[0-9]+%?$`

- **spec.cloudInstances.standbyHolder**
  object

  Amount of reserved resources.

  Used to determine whether to order overprovisioned nodes.

  - **spec.cloudInstances.standbyHolder.notHeldResources**
    Deprecated
    object

    Deprecated: the parameter is no longer used. Use the `overprovisioningRate` parameter.

    Describes the resources that will not be held (consumed) by the standby holder.

    - **spec.cloudInstances.standbyHolder.notHeldResources.cpu**
      integer or string

      Describes the amount of CPU that will not be held by standby holder on Nodes

from this NodeGroup.

The value can be an absolute number of cpus (for example, 2) as well as a milli representation (for example, 1500m).

Pattern: `^[0-9]+m?$`

- **spec.cloudInstances.standbyHolder.notHeldResources.memory**
  integer or string

  Describes the amount of memory that will not be held by standby holder on Nodes from this NodeGroup.

  The value can be an absolute number of bytes (for example, 128974848) as well as a fixed-point number using one of memory suffixes: G, Gi, M, Mi.

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **spec.cloudInstances.standbyHolder.overprovisioningRate**
  integer

  Percentage of reserved resources calculated from the capacity of a node of a `NodeGroup`.

  Default: `50`

  Allowed values: `1 <= X <= 80`

- **spec.cloudInstances.zones**
  array of strings

  List of availability zones to create instances in.

  The default value depends on the cloud provider selected and usually corresponds to all zones of the region being used.

  Example:

```
zones:
- Helsinki
- Espoo
- Tampere
```

- **spec.cri**
  object

  Container runtime parameters.

  - **spec.cri.containerd**
    object

    Containerd runtime parameters.

    If used, `cri.type` must be set to `Containerd`.

    - **spec.cri.containerd.maxConcurrentDownloads**
      integer

      Set the max concurrent downloads for each pull.

      Default: `3`

  - **spec.cri.containerdV2**
    object

    ContainerdV2 runtime parameters.

    If used, `cri.type` must be set to `ContainerdV2`.

    When using `ContainerdV2`, `CgroupsV2` will be used (providing improved security and resource management).

    To use `ContainerdV2` as a container runtime, the nodes in the group must meet the following requirements:

    - Support for `CgroupsV2`.
    - Linux kernel version `5.8` or newer.
    - Systemd version `244` or newer.
    - Support for `erofs` kernel module.
    - **spec.cri.containerdV2.maxConcurrentDownloads**
      integer

Set the max concurrent downloads for each pull.

Default: `3`

- **spec.cri.docker**
  Deprecated
  object

  Docker settings for nodes.

  - **spec.cri.docker.manage**
    boolean

    Enable Docker maintenance from bashible.

    Default: `true`

  - **spec.cri.docker.maxConcurrentDownloads**
    integer

    Set the max concurrent downloads for each pull.

    Default: `3`

- **spec.cri.notManaged**
  object

  Settings for not managed CRI for nodes.

  - **spec.cri.notManaged.criSocketPath**
    string

    Path to CRI socket.

- **spec.cri.type**
  string

  Container runtime type.

  If not specified, the `defaultCRI` parameter value of ClusterConfiguration is used.

  > **i** **Note.** Docker support is deprecated.

  Allowed values: `Docker`, `Containerd`, `ContainerdV2`, `NotManaged`

- **spec.disruptions**
  object

  Disruptions settings for nodes.

  Example:

```
disruptions:
  approvalMode: Automatic
  automatic:
    drainBeforeApproval: false
    windows:
    - from: '06:00'
      to: '08:00'
      days:
      - Tue
      - Sun
```

  - **spec.disruptions.approvalMode**
    string

    The approval mode for disruptive updates:

    - `Manual` — disable automatic disruption approval; the alert will be displayed if disruption is needed. Nodes are not drained automatically in `Manual` mode. If necessary, manually drain nodes before setting the `update.node.deckhouse.io/disruption-approved=` annotation. **Caution!** The `master` node group update mode must be `Manual` to avoid issues with draining.
    - `Automatic` — automatically approve disruption-involving updates. Nodes could be drained automatically before reboot, as configured by the drainBeforeApproval option.
    - `RollingUpdate` — in this mode, a **new** node with new settings will be created; then, the old node will be deleted. Available only for cloud nodes.

If the `RollingUpdate` mode is not used, when updating, the node is first drained and then updated (rebooted) and put back into operation (uncordoned). Note that in this case, the cluster must have sufficient resources to accommodate the load while the node being updated is unavailable. In the `RollingUpdate` mode, the node is **replaced** by the updated node, i.e., an extra node appears in the cluster for the duration of the update. In cloud infrastructures, the `RollingUpdate` mode is convenient, for example, if there are no resources in the cluster to temporarily host the load from the node being updated.

Default: `Automatic`

Allowed values: `Manual`, `Automatic`, `RollingUpdate`

- **spec.disruptions.automatic**
  object

  Additional parameters for the `Automatic` mode.

  - **spec.disruptions.automatic.drainBeforeApproval**
    boolean

    Drain Pods from the nodes before approving disruption.

    Caution! This setting ignores (nodes will be approved without draining Pods):

    - for the nodeGroup `master` with a single node;
    - for a single `ready` node in a nodeGroup picked out for Deckhouse placement.

    Default: `true`

  - **spec.disruptions.automatic.windows**
    array of objects

    Time windows for node disruptive updates.

    - **spec.disruptions.automatic.windows.days**
      array of strings

      Days of the week when node could be updated.

      Examples:

      ```
      days: Mon
      ```

      ```
      days: Wed
      ```

      - **Element of the array**
        string

        Day of the week.

        Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

    - **spec.disruptions.automatic.windows.from**
      string

      Required value

      Start time of disruptive update window (UTC timezone).

      Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

      Example:

      ```
      from: '13:00'
      ```

    - **spec.disruptions.automatic.windows.to**
      string

      Required value

      End time of disruptive update window (UTC timezone).

      Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

      Example:

      ```
      to: '18:30'
      ```

- **spec.disruptions.rollingUpdate**

object

Additional parameters for the `RollingUpdate` mode.

- **spec.disruptions.rollingUpdate.windows**
  array of objects

  Time windows for node disruptive updates.

  - **spec.disruptions.rollingUpdate.windows.days**
    array of strings

    Days of the week when node could be updated.

    Examples:

    ```
    days: Mon
    ```

    ```
    days: Wed
    ```

    - **Element of the array**
      string

      Day of the week.

      Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

  - **spec.disruptions.rollingUpdate.windows.from**
    string

    Required value

    Start time of disruptive update window (UTC timezone).

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    from: '13:00'
    ```

  - **spec.disruptions.rollingUpdate.windows.to**
    string

    Required value

    End time of disruptive update window (UTC timezone).

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    to: '18:30'
    ```

- **spec.fencing**
  object

  Enable the fencing-controller for a node group.

  - **spec.fencing.mode**
    string

    Required value

    Specifies the operating mode of the fencing-controller.

    Supported modes:

    - `Watchdog`: Uses the watchdog kernel module to automatically reboot nodes that lose connectivity with the cluster. In this mode, the fencing-agent also monitors node availability in the cluster.

    When `fencing` is enabled, the NodeGroupConfiguration script loads the `softdog` kernel module with the `soft margin` parameter set to the value of `spec.fencing.watchdog.timeout` (60 seconds by default) and `soft_panic=1`.

    How the fencing-agent works:

    - The agent opens the `/dev/watchdog` device and sets the `node-manager.deckhouse.io/fencing-enabled` label on the node, as well as the `node-`

`manager.deckhouse.io/fencing-mode` label with the value `Watchdog`.
- The fencing-agent joins a gossip cluster (`memberlist`) with agents from the same node group.
- The agent regularly checks quorum (availability of the majority of nodes in the group):
- If quorum is lost (`quorum=false`), the agent performs an additional check of Kubernetes API availability.
- If quorum is present or the Kubernetes API is reachable, the agent resets the watchdog timer.
- If neither quorum nor access to the Kubernetes API is available, the watchdog is no longer reset, the timer expires, and a kernel panic occurs.
- The agent respects maintenance annotations `node-manager.deckhouse.io/fencing-disable`, `update.node.deckhouse.io/approved`, and `update.node.deckhouse.io/disruption-approved`, temporarily disabling the watchdog during planned operations.

When `fencing` is enabled, automatic node reboot after a kernel panic is disabled at the OS level. This prevents a node from returning in an undefined state before it is properly handled (manually for static nodes, or via the cloud-provider-controller for cloud nodes).

The fencing-controller takes the node type (`node.deckhouse.io/type`) into account when handling a failed node:

- For cloud nodes (`CloudEphemeral`, `CloudPermanent`), the controller deletes the Node object, after which the cloud-provider-controller recreates the corresponding virtual machine.
- For static nodes (`Static`, `CloudStatic`), the controller only evicts workloads while keeping the Node object in the cluster.

For details on how fencing handles different node types, see FAQ.

Allowed values: `Watchdog`

- **spec.fencing.watchdog**
  object
  - **spec.fencing.watchdog.timeout**
    string

    Required value

    Watchdog timeout duration. Should be specified in seconds, minutes or hours.

    Default: `60s`

    Pattern: `^[0-9]+[smh]{1}$`

- **spec.gpu**
  object

  Available in editions: EE

  GPU parameters.

  - **spec.gpu.exclusive**
    object

    Without GPU sharing.

  - **spec.gpu.mig**
    object

    `MIG` sharing strategy parameters.

    - **spec.gpu.mig.customConfigs**
      array of objects

      Custom MIG partitioning by GPU index.

      Used only when `partedConfig: custom` is set.

      Represents an array of devices (a map keyed by `index`), where each entry refers to a GPU by its index and describes the slices to be created.

      Examples:

      ```
      customConfigs:
        profile: 3g.40gb
        count: 2
      ```

```
customConfigs:
  profile: 1g.10gb.me
  count: 1
```

- **spec.gpu.mig.customConfigs.index**
  integer

  Required value

  GPU index on the node.

  Allowed values: `0 <= X`

- **spec.gpu.mig.customConfigs.slices**
  array of objects

  Required value

  List of MIG slices to create.

  - **spec.gpu.mig.customConfigs.slices.count**
    integer

    Required value

    Number of slices in the selected profile.

    Default: `1`

    Allowed values: `1 <= X <= 7`

  - **spec.gpu.mig.customConfigs.slices.profile**
    string

    Required value

    MIG profile name.

    Pattern: `^[0-9]+g\.[0-9]+gb(\.me)?$`

    Examples:

    ```
    profile: 3g.40gb
    ```

    ```
    profile: 1g.10gb
    ```

    ```
    profile: 1g.10gb.me
    ```

- **spec.gpu.mig.partedConfig**
  string

  MIG configuration name.

  Standard `all-*` profiles are supported (the same partitioning applied to all GPUs), as well as the `custom` value, which allows defining partitioning individually for each GPU.

  Default: `all-1g.5gb`

- **spec.gpu.sharing**
  string

  GPU sharing strategy:

  - `Exclusive` — no sharing (each GPU is fully allocated to a single Pod).
  - `TimeSlicing` — time-sliced sharing (multiple Pods can share one GPU concurrently by splitting GPU time).
  - `MIG` — use NVIDIA Multi-Instance GPU (MIG) to split each GPU into multiple instances (if supported by hardware).

  Allowed values: `TimeSlicing`, `MIG`, `Exclusive`

- **spec.gpu.timeSlicing**
  object

  `TimeSlicing` sharing strategy parameters.

  - **spec.gpu.timeSlicing.partitionCount**
    number

Count of partition per GPU.

Default: `4`

Allowed values: `1 <= X <= 100`

- **spec.kubelet**
  object

  Kubelet settings for nodes.

  - **spec.kubelet.containerLogMaxFiles**
    integer

    How many rotated log files to store before deleting them.

    Default: `4`

    Allowed values: `1 <= X <= 20`

  - **spec.kubelet.containerLogMaxSize**
    string

    Maximum log file size before it is rotated.

    Default: `50Mi`

    Pattern: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

  - **spec.kubelet.maxPods**
    integer

    Set the max count of pods per node for given `NodeGroup`. When left unspecified, the system automatically determines this limit according to the value of the podSubnetNodeCIDRPrefix ClusterConfiguration parameter:

    - `120` pods for `podSubnetNodeCIDRPrefix` ≥ `24`;
    - `250` pods for `podSubnetNodeCIDRPrefix` = `23`;
    - `500` pods for `podSubnetNodeCIDRPrefix` = `22`;
    - `1000` pods for `podSubnetNodeCIDRPrefix` ≤ `21`.

  - **spec.kubelet.memorySwap**
    object

    Configuration of swap memory usage for kubelet.

    - **spec.kubelet.memorySwap.limitedSwap**
      object

      Configuration for the limited swap mode. Required when `swapBehavior` is set to `LimitedSwap`.

      - **spec.kubelet.memorySwap.limitedSwap.size**
        string

        Required value

        Size of the swap file to create, specified in GB (for example, “2G”, “4G”).

        Pattern: `^\d+G$`

        Example:

        ```
        size: 1G
        ```

    - **spec.kubelet.memorySwap.swapBehavior**
      string

      Defines how swap memory is handled on the node:

      - `NoSwap`: Swap is disabled. This is the traditional Kubernetes behavior.
      - `LimitedSwap`: Kubelet can use swap memory up to the limit specified in `limitedSwap.size`.

      For details on swap memory usage on nodes, refer to Kubernetes documentation.

      Allowed values: `NoSwap`, `LimitedSwap`

    - **spec.kubelet.memorySwap.swappiness**
      integer

      Defines the kernel’s tendency to use the swap.

- `0`: Minimal swap usage (only to avoid OOM).
- `60`: Default balanced behavior.
- `100`: Aggressive swap usage.

This value is set via `vm.swappiness` sysctl parameter.

Default: `60`

Allowed values: `0 <= X <= 100`

- **spec.kubelet.resourceReservation**
  object

  Management of resource reservation for system daemons on a node.

  More info in the Kubernetes documentation.

  - **spec.kubelet.resourceReservation.mode**
    string

    Defines the system resource reservation mode:

    - `Off` — disables resource reservation.
    - `Auto` — reserves resources based on the Node capacity.
    - `Static` — reserves resources based on the values provided by the user via the `static` parameter.

    Note that currently we do not use a dedicated group `cgroup` for resource reservation (`-system-reserved-cgroup` is not used).

    Default: `Auto`

  - **spec.kubelet.resourceReservation.static**
    object

    Resource reservation parameters for the 'Static' mode.

    - **spec.kubelet.resourceReservation.static.cpu**
      integer or string

      Pattern: `\d+[m]`

    - **spec.kubelet.resourceReservation.static.ephemeralStorage**
      integer or string

      Pattern: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

    - **spec.kubelet.resourceReservation.static.memory**
      integer or string

      Pattern: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

- **spec.kubelet.rootDir**
  string

  Directory path for managing kubelet files (volume mounts,etc).

  Default: `/var/lib/kubelet`

- **spec.kubelet.topologyManager**
  object

  Controls a topology manager.

  The topology manager allows the Kubernetes scheduler to take resource topology data into account when scheduling pods.

  In a cluster with NUMA nodes, the topology manager can improve application performance by aligning workloads with NUMA nodes that are more resource-efficient.

  For more information, see the Kubernetes documentation.

  - **spec.kubelet.topologyManager.enabled**
    boolean

    Enables Topology Management on a node.

    If the topology manager is enabled, strict resource allocation modes are activated for containers in Guaranteed pods. The following parameter values are set in the kubelet configuration:

    - `cpuManagerPolicy: static`: The kubelet exclusively assigns whole physical cores to a container with an integer CPU request.

- `memoryManagerPolicy: Static`: The kubelet assigns the container's memory to a minimum number of NUMA nodes (preferably one).

Default: `false`

- **spec.kubelet.topologyManager.policy**
  string

  Resource/topology alignment policy to use when scheduling a pod.

  There are four supported policies:

  - `None` — no resource alignment (default).
  - `BestEffort` — a preferred NUMA node is defined for each container. If there are no preferred nodes for the pod available, it will be scheduled to a node without taking topology manager data into account, while the already running pod will keep running.
  - `Restricted` — a preferred NUMA node is defined for each pod container. If there are no preferred nodes available, a new pod will not be scheduled to a node while an already running pod will be stopped.
  - `SingleNumaNode` — a preferred NUMA node is defined for each pod container. If there is no NUMA node available that suits **all pod containers**, a new pod will not be scheduled while an already running pod will be stopped.

  More information can be found in the Kubernetes documentation.

  Default: `None`

  Allowed values: `None`, `BestEffort`, `Restricted`, `SingleNumaNode`

- **spec.kubelet.topologyManager.scope**
  string

  Defines granularity with which resources are aligned:

  - `Container`. The preferred resources are computed for a container. The computation does not take into account that multiple containers may belong to the same pod.
  - `Pod`. The preferred resources are computed for the pod. All containers in the pod are treated as a whole in the computation.

  Default: `Container`

  Allowed values: `Container`, `Pod`

- **spec.nodeDrainTimeoutSecond**
  integer

  Maximum duration (in seconds) for which the controller attempts to reschedule pods to a new machine during node draining.

- **spec.nodeTemplate**
  object

  Specification of some of the fields that will be maintained in all nodes of the group.

  Example:

```
nodeTemplate:
  labels:
    environment: production
    app: warp-drive-ai
  annotations:
    ai.fleet.com/discombobulate: 'true'
  taints:
  - effect: NoExecute
    key: ship-class
    value: frigate
```

  - **spec.nodeTemplate.annotations**
    object

    Similar to the standard `metadata.annotations` field.

    Example:

```
annotations:
  ai.fleet.com/discombobulate: 'true'
```

- **spec.nodeTemplate.labels**
  object

  Similar to the standard `metadata.labels` field.

  Example:

  ```
  labels:
    environment: production
    app: warp-drive-ai
  ```

- **spec.nodeTemplate.taints**
  array of objects

  Similar to the `.spec.taints` field of the Node object.

  > **i** Available fields: `effect`, `key`, and `values`.

  Example:

  ```
  taints:
  - effect: NoExecute
    key: ship-class
    value: frigate
  ```

  - **spec.nodeTemplate.taints.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **spec.nodeTemplate.taints.key**
    string
  - **spec.nodeTemplate.taints.value**
    string

- **spec.nodeType**
  string

  Required value

  The type of nodes this group provides:

  - `CloudEphemeral` — nodes for this group will be automatically created (and deleted) in the cloud of the specified cloud provider;
  - `CloudPermanent` — nodes from ProviderClusterConfiguration will be created via dhctl;
  - `CloudStatic` — a static node (created manually or using any external tools) hosted in the cloud integrated with one of the cloud providers. This node has the CSI running, and it is managed by the cloud-controller-manager: the `Node` object automatically gets the information about the zone and region based on the cloud data; if a node gets deleted from the cloud, its corresponding Node object will be deleted in Kubernetes;
  - `Static` — a static node hosted on a bare metal or virtual machine. The cloud-controller-manager does not manage the node even if one of the cloud providers is enabled.

  Allowed values: `CloudEphemeral`, `CloudPermanent`, `CloudStatic`, `Static`

- **spec.operatingSystem**
  object

  Operating System settings for nodes.

  - **spec.operatingSystem.manageKernel**
    Deprecated
    boolean

    This parameter has no effect. Earlier, it enabled kernel maintenance on behalf of bashible.

    Default: `true`

- **spec.staticInstances**
  object

  Parameter for provisioning static machines to the cluster.

- **spec.staticInstances.count**
  integer

  Target number of StaticInstance resources (matching the `labelSelector`) that DKP bootstraps and adds into the cluster.

  In a NodeGroup with Static or CloudStatic nodes, this parameter sets the expected number of nodes. DKP uses this value for state control and automation purposes.

  > **Warning.** Set this parameter explicitly. The default value is `0`, which means DKP won't bootstrap any StaticInstance resource or add any node to the group unless this value is modified. To add static nodes, specify the required number here (greater than `0`).

  Default: `0`

  Allowed values: `0 <= X`

- **spec.staticInstances.labelSelector**
  object

  A label selector is a label query over a set of `staticInstances` resources. The result of matchLabels and matchExpressions are ANDed. An empty label selector matches all objects. A null label selector matches no objects.

  - **spec.staticInstances.labelSelector.matchExpressions**
    array of objects

    A list of label selector requirements. The requirements are ANDed.

    A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

    - **spec.staticInstances.labelSelector.matchExpressions.key**
      string

      key is the label key that the selector applies to.

    - **spec.staticInstances.labelSelector.matchExpressions.operator**
      string

      operator represents a key's relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

    - **spec.staticInstances.labelSelector.matchExpressions.values**
      array of strings

      values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

      - **Element of the array**
        string

        Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

        Length: `1..63`

  - **spec.staticInstances.labelSelector.matchLabels**
    object

    matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is "key", the operator is "In", and the values array contains only "value". The requirements are ANDed.

- **spec.update**
  object
  - **spec.update.maxConcurrent**
    integer or string

    Maximum number of concurrently updating nodes.

    Can be set as absolute count or as a percent of total nodes.

    Default: `1`

    Pattern: `^[1-9][0-9]*%?$`

Describes the runtime parameters of the node group.

- **metadata**
  object
  - **metadata.name**
    string

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

    Maximum length: `42`

- **status**
  object
  - **status.conditionSummary**
    object
    - **status.conditionSummary.ready**
      string

      Status of the condition summary.

      Allowed values: `True`, `False`

    - **status.conditionSummary.statusMessage**
      string

      Status message about group handling.

  - **status.desired**
    integer

    Number of desired machines in the group.

  - **status.error**
    string

    Error message about possible problems with the group handling.

  - **status.instances**
    integer

    Number of instances (in any state) in the group.

  - **status.kubernetesVersion**
    string

    The current version of kubernetes on the nodes, or the version to which the nodes will be upgraded.

  - **status.lastMachineFailures**
    array of objects
    - **status.lastMachineFailures.lastOperation**
      object
      - **status.lastMachineFailures.lastOperation.description**
        string

        Last operation’s description.

      - **status.lastMachineFailures.lastOperation.lastUpdateTime**
        string

        Timestamp of last status update for operation.

      - **status.lastMachineFailures.lastOperation.state**
        string

        Machine’s operation state.

      - **status.lastMachineFailures.lastOperation.type**
        string

        Type of operation.

    - **status.lastMachineFailures.name**
      string

      Machine’s name.

    - **status.lastMachineFailures.ownerRef**
      string

      Machine owner’s name.

    - **status.lastMachineFailures.providerID**
      string

Machine’s ProviderID.

- **status.max**
  integer

  Maximum amount of instances in the group.

- **status.min**
  integer

  Minimal amount of instances in the group.

- **status.nodes**
  integer

  Number of Kubernetes nodes (in any state) in the group.

- **status.ready**
  integer

  Number of ready Kubernetes nodes in the group.

- **status.standby**
  integer

  Number of overprovisioned instances in the group.

- **status.upToDate**
  integer

  Number of up-to-date nodes in the group.

- **spec**
  object

  Required value

  - **spec.chaos**
    object

    Chaos monkey settings.

    Example:

    ```
    chaos:
      mode: DrainAndDelete
      period: 24h
    ```

    - **spec.chaos.mode**
      string

      The chaos monkey mode:

      - `DrainAndDelete` — drains and deletes a node when triggered;
      - `Disabled` — leaves this NodeGroup intact.

      Default: `Disabled`

      Allowed values: `Disabled`, `DrainAndDelete`

    - **spec.chaos.period**
      string

      The time interval to use for the chaos monkey (can be specified in the Go format).

      Default: `6h`

      Pattern: `^[0-9]+[mh]{1}$`

  - **spec.cloudInstances**
    object

    Parameter for provisioning the cloud-based VMs.

    > **i** **Caution!** Can only be used together with `nodeType: CloudEphemeral`.

    - **spec.cloudInstances.classReference**
      object

      Required value

The reference to the `InstanceClass` object. It is unique for each `cloud-provider-*` module.

- **spec.cloudInstances.classReference.kind**
  string

  The object type (e.g., `OpenStackInstanceClass`). The object type is specified in the documentation of the corresponding `cloud-provider-` module.

  Allowed values: `OpenStackInstanceClass`, `GCPInstanceClass`, `VsphereInstanceClass`, `AWSInstanceClass`, `YandexInstanceClass`, `AzureInstanceClass`, `VCDInstanceClass`, `ZvirtInstanceClass`, `DynamixInstanceClass`, `HuaweiCloudInstanceClass`, `DVPInstanceClass`

- **spec.cloudInstances.classReference.name**
  string

  The name of the required `InstanceClass` object (e.g., `finland-medium`).

- **spec.cloudInstances.maxPerZone**
  integer

  Required value

  The maximum number of instances for the group in each zone.

  This value is used as the upper bound in cluster-autoscaler.

  Allowed values: `0 <= X`

- **spec.cloudInstances.maxSurgePerZone**
  integer

  The maximum number of instances to rollout simultaneously in the group in each zone.

  Default: `1`

  Allowed values: `0 <= X`

- **spec.cloudInstances.maxUnavailablePerZone**
  integer

  The maximum number of unavailable instances (during rollout) in the group in each zone.

  Default: `0`

  Allowed values: `0 <= X`

- **spec.cloudInstances.minPerZone**
  integer

  Required value

  The minimum number of instances for the group in each zone.

  This value is used in the `MachineDeployment` object and as a lower bound in cluster-autoscaler.

  Allowed values: `0 <= X`

- **spec.cloudInstances.standby**
  integer or string

  The summary number of overprovisioned nodes for this `NodeGroup` in all zones.

  An overprovisioned node is a cluster node on which resources are reserved that are available at any time for scaling. The presence of such a node allows the cluster autoscaler not to wait for node initialization (which may take several minutes), but to immediately place a load on it.

  The value can be an absolute number (for example, 2) or a percentage of desired nodes (for example, 10%). If a percentage is specified, the absolute number is calculated based on the percentage of the maximum number of nodes (the maxPerZone parameter) rounded down, but not less than one.

  Pattern: `^[0-9]+%?$`

- **spec.cloudInstances.standbyHolder**
  object

  Amount of reserved resources.

  Used to determine whether to order overprovisioned nodes.

- **spec.cloudInstances.standbyHolder.notHeldResources**
  object

  Describes the resources that will not be held (consumed) by the standby holder.

  - **spec.cloudInstances.standbyHolder.notHeldResources.cpu**
    integer or string

    Describes the amount of CPU that will not be held by standby holder on Nodes from this NodeGroup.

    The value can be an absolute number of cpus (for example, 2) as well as a milli representation (for example, 1500m).

    Pattern: `^[0-9]+m?$`

  - **spec.cloudInstances.standbyHolder.notHeldResources.memory**
    integer or string

    Describes the amount of memory that will not be held by standby holder on Nodes from this NodeGroup.

    The value can be an absolute number of bytes (for example, 128974848) as well as a fixed-point number using one of memory suffixes: G, Gi, M, Mi.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **spec.cloudInstances.zones**
  array of strings

  List of availability zones to create instances in.

  The default value depends on the cloud provider selected and usually corresponds to all zones of the region being used.

  Example:

  ```
  zones:
  - Helsinki
  - Espoo
  - Tampere
  ```

- **spec.cri**
  object

  Container runtime parameters.

  - **spec.cri.containerd**
    object

    Containerd runtime parameters.

    If used, `cri.type` must be set to `Containerd`.

    - **spec.cri.containerd.maxConcurrentDownloads**
      integer

      Set the max concurrent downloads for each pull.

      Default: `3`

  - **spec.cri.docker**
    object

    Docker settings for nodes.

    > **i** **Note!** the `Docker` is deprecated.

    - **spec.cri.docker.manage**
      boolean

      Enable Docker maintenance from bashible.

      Default: `true`

    - **spec.cri.docker.maxConcurrentDownloads**
      integer

      Set the max concurrent downloads for each pull.

Default: `3`

- **spec.cri.notManaged**
  object

  Settings for not managed CRI for nodes.

  - **spec.cri.notManaged.criSocketPath**
    string

    Path to CRI socket.

- **spec.cri.type**
  string

  Container runtime type.

  Value `defaultCRI` from the initial cluster configration (`cluster-configuration.yaml` parameter from the `d8-cluster-configuration` secret in the `kube-system` namespace) is used if not specified.

  > **i** **Note.** Docker support is deprecated.

  Allowed values: `Docker`, `Containerd`, `NotManaged`

- **spec.disruptions**
  object

  Disruptions settings for nodes.

  Example:

```
disruptions:
  approvalMode: Automatic
  automatic:
    drainBeforeApproval: false
    windows:
    - from: '06:00'
      to: '08:00'
      days:
      - Tue
      - Sun
```

  - **spec.disruptions.approvalMode**
    string

    The approval mode for disruptive updates:

    - `Manual` — disable automatic disruption approval; the alert will be displayed if disruption is needed. **Caution!** The `master` node group update mode must be `Manual` to avoid issues with draining.
    - `Automatic` — automatically approve disruption-involving updates.
    - `RollingUpdate` — in this mode, a **new** node with new settings will be created; then, the old node will be deleted. Available only for cloud nodes.

    If the `RollingUpdate` mode is not used, when updating, the node is first drained and then updated (rebooted) and put back into operation (uncordoned). Note that in this case, the cluster must have sufficient resources to accommodate the load while the node being updated is unavailable. In the `RollingUpdate` mode, the node is **replaced** by the updated node, i.e., an extra node appears in the cluster for the duration of the update. In cloud infrastructures, the `RollingUpdate` mode is convenient, for example, if there are no resources in the cluster to temporarily host the load from the node being updated.

    Default: `Automatic`

    Allowed values: `Manual`, `Automatic`, `RollingUpdate`

  - **spec.disruptions.automatic**
    object

    Additional parameters for the `Automatic` mode.

    - **spec.disruptions.automatic.drainBeforeApproval**
      boolean

      Drain Pods from the nodes before approving disruption.

Caution! This setting ignores (nodes will be approved without draining Pods):

- for the nodeGroup `master` with a single node;
- for a single `ready` node in a nodeGroup picked out for Deckhouse placement.

Default: `true`

- **spec.disruptions.automatic.windows**
  array of objects

  Time windows for node disruptive updates.

  - **spec.disruptions.automatic.windows.days**
    array of strings

    Days of the week when node could be updated.

    Examples:

    ```
    days: Mon
    ```

    ```
    days: Wed
    ```

    - **Element of the array**
      string

      Day of the week.

      Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

  - **spec.disruptions.automatic.windows.from**
    string

    Required value

    Start time of disruptive update window (UTC timezone).

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    from: '13:00'
    ```

  - **spec.disruptions.automatic.windows.to**
    string

    Required value

    End time of disruptive update window (UTC timezone).

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    to: '18:30'
    ```

- **spec.disruptions.rollingUpdate**
  object

  Additional parameters for the `RollingUpdate` mode.

  - **spec.disruptions.rollingUpdate.windows**
    array of objects

    Time windows for node disruptive updates.

    - **spec.disruptions.rollingUpdate.windows.days**
      array of strings

      Days of the week when node could be updated.

      Examples:

      ```
      days: Mon
      ```

      ```
      days: Wed
      ```

        - **Element of the array**
          string

          Day of the week.

          Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

      - **spec.disruptions.rollingUpdate.windows.from**
        string

        Required value

        Start time of disruptive update window (UTC timezone).

        Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

        Example:

        ```
        from: '13:00'
        ```

      - **spec.disruptions.rollingUpdate.windows.to**
        string

        Required value

        End time of disruptive update window (UTC timezone).

        Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

        Example:

        ```
        to: '18:30'
        ```

- **spec.kubelet**
  object

  Kubelet settings for nodes.

  - **spec.kubelet.containerLogMaxFiles**
    integer

    How many rotated log files to store before deleting them.

    > **WARNING!** This parameter does nothing if CRI type is `Docker`.

    Default: `4`

    Allowed values: `1 <= X <= 20`

  - **spec.kubelet.containerLogMaxSize**
    string

    Maximum log file size before it is rotated.

    > **WARNING!** This parameter does nothing if CRI type is `Docker`.

    Default: `50Mi`

    Pattern: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

  - **spec.kubelet.maxPods**
    integer

    Set the max count of pods per node for given `NodeGroup`. When left unspecified, the system automatically determines this limit according to the value of the podSubnetNodeCIDRPrefix ClusterConfiguration parameter:

    - `120` pods for `podSubnetNodeCIDRPrefix` ≥ `24`;
    - `250` pods for `podSubnetNodeCIDRPrefix` = `23`;
    - `500` pods for `podSubnetNodeCIDRPrefix` = `22`;
    - `1000` pods for `podSubnetNodeCIDRPrefix` ≤ `21`.
  - **spec.kubelet.rootDir**
    string

    Directory path for managing kubelet files (volume mounts,etc).

Default: `/var/lib/kubelet`

- **spec.nodeTemplate**
  object

  Specification of some of the fields that will be maintained in all nodes of the group.

  Example:

  ```
  nodeTemplate:
    labels:
      environment: production
      app: warp-drive-ai
    annotations:
      ai.fleet.com/discombobulate: 'true'
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
  ```

  - **spec.nodeTemplate.annotations**
    object

    Similar to the standard `metadata.annotations` field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: 'true'
    ```

  - **spec.nodeTemplate.labels**
    object

    Similar to the standard `metadata.labels` field.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

  - **spec.nodeTemplate.taints**
    array of objects

    Similar to the `.spec.taints` field of the Node object.

    > **i** Available fields: `effect`, `key`, and `values`.

    Example:

    ```
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
    ```

    - **spec.nodeTemplate.taints.effect**
      string

      Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

    - **spec.nodeTemplate.taints.key**
      string
    - **spec.nodeTemplate.taints.value**
      string

- **spec.nodeType**
  string

  Required value

  The type of nodes this group provides.

  - `Cloud` — nodes for this group will be automatically created (and deleted) in the cloud of the

specified cloud provider;
- `Static` — a static node hosted on a bare metal or virtual machine. The cloud-controller-manager does not manage the node even of one of the cloud providers is enabled;
- `Hybrid` — a static node (created manually or using any external tools) hosted in the cloud integrated with one of the cloud provider. This node has the CSI running, and it is managed by the cloud-controller-manager: the Node object automatically gets the information about the zone and region based on the cloud data; if a node gets deleted from the cloud, its corresponding Node object will be deleted in Kubernetes.

Allowed values: `Cloud`, `Static`, `Hybrid`

- **spec.operatingSystem**
  object

  Operating System settings for nodes.

  - **spec.operatingSystem.manageKernel**
    boolean

    Enable kernel maintenance from bashible.

    Default: `true`

Defines the runtime parameters of a node group.

- **metadata**
  object
  - **metadata.name**
    string

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?$`

    Maximum length: `42`
- **status**
  object
  - **status.conditionSummary**
    object
    - **status.conditionSummary.ready**
      string

      Status of the condition summary.

      Allowed values: `True`, `False`

    - **status.conditionSummary.statusMessage**
      string

      Status message about group handling.
  - **status.desired**
    integer

    Number of desired machines in the group.
  - **status.error**
    string

    Error message about possible problems with the group handling.
  - **status.instances**
    integer

    Number of instances (in any state) in the group.
  - **status.kubernetesVersion**
    string

    The current version of kubernetes on the nodes, or the version to which the nodes will be upgraded.
  - **status.lastMachineFailures**
    array of objects
    - **status.lastMachineFailures.lastOperation**
      object
      - **status.lastMachineFailures.lastOperation.description**
        string

        Last operation’s description.

      - **status.lastMachineFailures.lastOperation.lastUpdateTime**
        string

Timestamp of last status update for operation.

- **status.lastMachineFailures.lastOperation.state**
  string

  Machine’s operation state.

- **status.lastMachineFailures.lastOperation.type**
  string

  Type of operation.

- **status.lastMachineFailures.name**
  string

  Machine’s name.

- **status.lastMachineFailures.ownerRef**
  string

  Machine owner’s name.

- **status.lastMachineFailures.providerID**
  string

  Machine’s ProviderID.

- **status.max**
  integer

  Maximum amount of instances in the group.

- **status.min**
  integer

  Minimal amount of instances in the group.

- **status.nodes**
  integer

  Number of Kubernetes nodes (in any state) in the group.

- **status.ready**
  integer

  Number of ready Kubernetes nodes in the group.

- **status.standby**
  integer

  Number of overprovisioned instances in the group.

- **status.upToDate**
  integer

  Number of up-to-date nodes in the group.

- **spec**
  object

  Required value

  - **spec.chaos**
    object

    Chaos monkey settings.

    Example:

    ```
    chaos:
      mode: DrainAndDelete
      period: 24h
    ```

    - **spec.chaos.mode**
      string

      The chaos monkey mode:

      - `DrainAndDelete` — drains and deletes a node when triggered;
      - `Disabled` — leaves this NodeGroup intact.

      Default: `Disabled`

Allowed values: `Disabled`, `DrainAndDelete`

- **spec.chaos.period**
  string

  The time interval to use for the chaos monkey (can be specified in the Go format).

  Default: `6h`

  Pattern: `^[0-9]+[mh]{1}$`

- **spec.cloudInstances**
  object

  Parameter for provisioning the cloud-based VMs.

  > **Caution!** Can only be used together with `nodeType: CloudEphemeral`.

  - **spec.cloudInstances.classReference**
    object

    Required value

    The reference to the `InstanceClass` object. It is unique for each `cloud-provider-*` module.

    - **spec.cloudInstances.classReference.kind**
      string

      The object type (e.g., `OpenStackInstanceClass`). The object type is specified in the documentation of the corresponding `cloud-provider-` module.

      Allowed values: `OpenStackInstanceClass`, `GCPInstanceClass`, `VsphereInstanceClass`, `AWSInstanceClass`, `YandexInstanceClass`, `AzureInstanceClass`, `VCDInstanceClass`, `ZvirtInstanceClass`, `DynamixInstanceClass`, `HuaweiCloudInstanceClass`, `DVPInstanceClass`

    - **spec.cloudInstances.classReference.name**
      string

      The name of the required `InstanceClass` object (e.g., `finland-medium`).

  - **spec.cloudInstances.maxPerZone**
    integer

    Required value

    The maximum number of instances for the group in each zone.

    This value is used as the upper bound in cluster-autoscaler.

    With a value of 0, you need to set capacity for some `InstanceClass`. Get more details in the description of the necessary `InstanceClass`.

    Allowed values: `0 <= X`

  - **spec.cloudInstances.maxSurgePerZone**
    integer

    The maximum number of instances to rollout simultaneously in the group in each zone.

    Default: `1`

    Allowed values: `0 <= X`

  - **spec.cloudInstances.maxUnavailablePerZone**
    integer

    The maximum number of unavailable instances (during rollout) in the group in each zone.

    Default: `0`

    Allowed values: `0 <= X`

  - **spec.cloudInstances.minPerZone**
    integer

    Required value

    The minimum number of instances for the group in each zone.

This value is used in the `MachineDeployment` object and as a lower bound in cluster-autoscaler.

Allowed values: `0 <= X`

- **spec.cloudInstances.standby**
integer or string

The summary number of overprovisioned nodes for this `NodeGroup` all zones.

An overprovisioned node is a cluster node on which resources are reserved that are available at any time for scaling. The presence of such a node allows the cluster autoscaler not to wait for node initialization (which may take several minutes), but to immediately place a load on it.

The value can be an absolute number (for example, 2) or a percentage of desired nodes (for example, 10%). If a percentage is specified, the absolute number is calculated based on the percentage of the maximum number of nodes (the maxPerZone parameter) rounded down, but not less than one.

Pattern: `^[0-9]+%?$`

- **spec.cloudInstances.standbyHolder**
object

Amount of reserved resources.

Used to determine whether to order overprovisioned nodes.

  - **spec.cloudInstances.standbyHolder.notHeldResources**
  object

  Describes the resources that will not be held (consumed) by the standby holder.

    - **spec.cloudInstances.standbyHolder.notHeldResources.cpu**
    integer or string

    Describes the amount of CPU that will not be held by standby holder on Nodes from this NodeGroup.

    The value can be an absolute number of cpus (for example, 2) as well as a milli representation (for example, 1500m).

    Pattern: `^[0-9]+m?$`

    - **spec.cloudInstances.standbyHolder.notHeldResources.memory**
    integer or string

    Describes the amount of memory that will not be held by standby holder on Nodes from this NodeGroup.

    The value can be an absolute number of bytes (for example, 128974848) as well as a fixed-point number using one of memory suffixes: G, Gi, M, Mi.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|K|Ei|Pi|Ti|Gi|Mi|Ki)?$`

- **spec.cloudInstances.zones**
array of strings

List of availability zones to create instances in.

The default value depends on the cloud provider selected and usually corresponds to all zones of the region being used.

Example:

```
zones:
- Helsinki
- Espoo
- Tampere
```

- **spec.cri**
object

Container runtime parameters.

  - **spec.cri.containerd**
  object

  Containerd runtime parameters.

If used, `cri.type` must be set to `Containerd`.

- **spec.cri.containerd.maxConcurrentDownloads**
  integer

  Set the max concurrent downloads for each pull.

  Default: `3`

- **spec.cri.type**
  string

  Container runtime type.

  If not specified, the `defaultCRI` parameter value of ClusterConfiguration is used.

  > **Note.** Docker support is deprecated.

  Allowed values: `Docker`, `Containerd`, `NotManaged`

- **spec.disruptions**
  object

  Disruptions settings for nodes.

  Example:

```
disruptions:
  approvalMode: Automatic
  automatic:
    drainBeforeApproval: false
    windows:
    - from: '06:00'
      to: '08:00'
      days:
      - Tue
      - Sun
```

  - **spec.disruptions.approvalMode**
    string

    The approval mode for disruptive updates:

    - `Manual` — disable automatic disruption approval; the alert will be displayed if disruption is needed. **Caution!** The `master` node group update mode must be `Manual` to avoid issues with draining.
    - `Automatic` — automatically approve disruption-involving updates.
    - `RollingUpdate` — in this mode, a **new** node with new settings will be created; then, the old node will be deleted. Available only for cloud nodes.

    If the `RollingUpdate` mode is not used, when updating, the node is first drained and then updated (rebooted) and put back into operation (uncordoned). Note that in this case, the cluster must have sufficient resources to accommodate the load while the node being updated is unavailable. In the `RollingUpdate` mode, the node is **replaced** by the updated node, i.e., an extra node appears in the cluster for the duration of the update. In cloud infrastructures, the `RollingUpdate` mode is convenient, for example, if there are no resources in the cluster to temporarily host the load from the node being updated.

    Default: `Automatic`

    Allowed values: `Manual`, `Automatic`, `RollingUpdate`

  - **spec.disruptions.automatic**
    object

    Additional parameters for the `Automatic` mode.

    - **spec.disruptions.automatic.drainBeforeApproval**
      boolean

      Drain Pods from the nodes before approving disruption.

      Caution! This setting ignores (nodes will be approved without draining Pods):

      - for the nodeGroup `master` with a single node;
      - for a single `ready` node in a nodeGroup picked out for Deckhouse placement.

Default: `true`

- **spec.disruptions.automatic.windows**
  array of objects

  Time windows for node disruptive updates.

  - **spec.disruptions.automatic.windows.days**
    array of strings

    Days of the week when node could be updated.

    Examples:

    ```
    days: Mon
    ```

    ```
    days: Wed
    ```

    - **Element of the array**
      string

      Day of the week.

      Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

  - **spec.disruptions.automatic.windows.from**
    string

    Required value

    Start time of disruptive update window (UTC timezone).

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    from: '13:00'
    ```

  - **spec.disruptions.automatic.windows.to**
    string

    Required value

    End time of disruptive update window (UTC timezone).

    Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

    Example:

    ```
    to: '18:30'
    ```

- **spec.disruptions.rollingUpdate**
  object

  Additional parameters for the `RollingUpdate` mode.

  - **spec.disruptions.rollingUpdate.windows**
    array of objects

    Time windows for node disruptive updates.

    - **spec.disruptions.rollingUpdate.windows.days**
      array of strings

      Days of the week when node could be updated.

      Examples:

      ```
      days: Mon
      ```

      ```
      days: Wed
      ```

      - **Element of the array**
        string

        Day of the week.

Allowed values: `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`

- **spec.disruptions.rollingUpdate.windows.from**
  string

  Required value

  Start time of disruptive update window (UTC timezone).

  Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

  Example:

  ```
  from: '13:00'
  ```

- **spec.disruptions.rollingUpdate.windows.to**
  string

  Required value

  End time of disruptive update window (UTC timezone).

  Pattern: `^(?:\d|[01]\d|2[0-3]):[0-5]\d$`

  Example:

  ```
  to: '18:30'
  ```

- **spec.docker**
  object

  Docker settings for nodes.

  If used, `cri.type` must be set to `Docker`.

  > **Note!** the `Docker` is deprecated.

  - **spec.docker.manage**
    boolean

    Enable Docker maintenance from bashible.

    Default: `true`

  - **spec.docker.maxConcurrentDownloads**
    integer

    Set the max concurrent downloads for each pull.

    Default: `3`

- **spec.kubelet**
  object

  Kubelet settings for nodes.

  - **spec.kubelet.containerLogMaxFiles**
    integer

    How many rotated log files to store before deleting them.

    > **WARNING!** This parameter does nothing if CRI type is `Docker`.

    Default: `4`

    Allowed values: `1 <= X <= 20`

  - **spec.kubelet.containerLogMaxSize**
    string

    Maximum log file size before it is rotated.

    > **WARNING!** This parameter does nothing if CRI type is `Docker`.

Default: `50Mi`

Pattern: `\d+[Ei|Pi|Ti|Gi|Mi|Ki|E|P|T|G|M|k|m]`

- **spec.kubelet.maxPods**
  integer

  Set the max count of pods per node for given `NodeGroup`. When left unspecified, the system automatically determines this limit according to the value of the podSubnetNodeCIDRPrefix ClusterConfiguration parameter:

  - `120` pods for `podSubnetNodeCIDRPrefix` ≥ `24`;
  - `250` pods for `podSubnetNodeCIDRPrefix` = `23`;
  - `500` pods for `podSubnetNodeCIDRPrefix` = `22`;
  - `1000` pods for `podSubnetNodeCIDRPrefix` ≤ `21`.
- **spec.kubelet.rootDir**
  string

  Directory path for managing kubelet files (volume mounts,etc).

  Default: `/var/lib/kubelet`

- **spec.kubernetesVersion**
  string

  The desired minor version of Kubernetes.

  By default, it corresponds to the version selected for the cluster globally (see installation documentation) or to the current version of the control plane (if the global version is not defined).

  Allowed values: `1.31`, `1.32`, `1.33`, `1.34`, `1.35`

  Example:

  ```
  kubernetesVersion: '1.32'
  ```

- **spec.nodeTemplate**
  object

  Specification of some of the fields that will be maintained in all nodes of the group.

  Example:

  ```
  nodeTemplate:
    labels:
      environment: production
      app: warp-drive-ai
    annotations:
      ai.fleet.com/discombobulate: 'true'
    taints:
    - effect: NoExecute
      key: ship-class
      value: frigate
  ```

  - **spec.nodeTemplate.annotations**
    object

    Similar to the standard `metadata.annotations` field.

    Example:

    ```
    annotations:
      ai.fleet.com/discombobulate: 'true'
    ```

  - **spec.nodeTemplate.labels**
    object

    Similar to the standard `metadata.labels` field.

    Example:

    ```
    labels:
      environment: production
      app: warp-drive-ai
    ```

- **spec.nodeTemplate.taints**
  array of objects

  Similar to the `.spec.taints` field of the Node object.

  > **i** Available fields: `effect`, `key`, and `values`.

  Example:

  ```
  taints:
  - effect: NoExecute
    key: ship-class
    value: frigate
  ```

  - **spec.nodeTemplate.taints.effect**
    string

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **spec.nodeTemplate.taints.key**
    string
  - **spec.nodeTemplate.taints.value**
    string

- **spec.nodeType**
  string

  Required value

  The type of nodes this group provides.

  - `Cloud` — nodes for this group will be automatically created (and deleted) in the cloud of the specified cloud provider;
  - `Static` — a static node hosted on a bare metal or virtual machine. The cloud-controller-manager does not manage the node even of one of the cloud providers is enabled;
  - `Hybrid` — a static node (created manually or using any external tools) hosted in the cloud integrated with one of the cloud provider. This node has the CSI running, and it is managed by the cloud-controller-manager: the Node object automatically gets the information about the zone and region based on the cloud data; if a node gets deleted from the cloud, its corresponding Node object will be deleted in Kubernetes.

  Allowed values: `Cloud`, `Static`, `Hybrid`

- **spec.operatingSystem**
  Deprecated
  object

  Operating System settings for nodes.

  - **spec.operatingSystem.manageKernel**
    Deprecated
    boolean

    Enable kernel maintenance from bashible.

    Default: `true`

- **spec.static**
  object

  Static node parameters

  - **spec.static.internalNetworkCIDRs**
    array of strings

    Subnet CIDR

NodeGroupConfiguration

Short names: ngc

Scope: Cluster
Version: v1alpha1

Executes bash scripts on nodes.

Read more in the module documentation.

- **spec**
  object

  Required value

  - **spec.bundles**
    array of strings

    Required value

    Bundles for step execution. You can set `'*'` for selecting all bundles.

    Examples:

    ```
    bundles:
    - ubuntu-lts
    - centos-7
    ```

    ```
    bundles:
    - ubuntu-lts
    ```

    ```
    bundles:
    - "*"
    ```

    - **Element of the array**
      string

      Allowed values: `ubuntu-lts`, `centos`, `debian`, `redos`, `rosa`, `astra`, `altlinux`, `opensuse`, `*`

  - **spec.content**
    string

    Required value

    A bash script that does the same things you would do in a configuration step.

    You can use Go Template to generate a script.

    The list of parameters available for use in templates can be retrieved from the `bashible-apiserver-context` Secret as follows:

    ```
    d8 k -n d8-cloud-instance-manager get secrets bashible-apiserver-context -o
    jsonpath='{.data.input\.yaml}'| base64 -d
    ```

    For example:

    ```
    {{- range .nodeUsers }}
    echo 'Tuning environment for user {{ .name }}'
    # Some code for tuning user environment
    {{- end }}
    ```

    You can also use the pre-defined bashbooster commands in the script. For example:

    ```
    bb-event-on 'bb-package-installed' 'post-install'
    post-install() {
      bb-log-info "Setting reboot flag due to kernel was updated"
      bb-flag-set reboot
    }
    ```

    Example of using the proxy parameters of ClusterConfiguration resource:

    ```
    bb-set-proxy
    curl -I https://google.com
    bb-unset-proxy
    ```

```
test-function-with-proxy() {
  bb-set-proxy
  trap bb-unset-proxy RETURN
  curl -I https://google.com
}
test-function-with-proxy
```

- **spec.nodeGroups**
  array of strings

  Required value

  List of NodeGroups to apply the step for. You can set `'*'` for selecting all NodeGroups.

  Examples:

  ```
  nodeGroups:
  - master
  - worker
  ```

  ```
  nodeGroups:
  - worker
  ```

  ```
  nodeGroups:
  - "*"
  ```

- **spec.weight**
  integer

  Order of the step execution.

  Default: `100`

NodeUser

Scope: Cluster

- v1
- v1alpha1

Defines the linux users to create on all nodes.

The user's home directory is created in the `/home/deckhouse/` directory.

- **spec**
  object

  Required value

  - **spec.extraGroups**
    array of strings

    Node user additional system groups.

    Examples:

    ```
    extraGroups:
    - docker
    ```

    ```
    extraGroups:
    - docker
    - ftp
    ```

  - **spec.isSudoer**
    boolean

    Persistence of node user in sudo group.

    Default: `false`

    Example:

```
isSudoer: true
```

- **spec.nodeGroups**
  array of strings

  List of NodeGroups to apply the user for.

  Default: `["*"]`

  Examples:

  ```
  nodeGroups:
  - master
  - worker
  ```

  ```
  nodeGroups:
  - worker
  ```

  ```
  nodeGroups:
  - "*"
  ```

- **spec.passwordHash**
  string

  Hashed user password.

  The format corresponds to the password hashes in `/etc/shadow`. You can get it using the following command: `openssl passwd -6`.

  Example:

  ```
  passwordHash: "$2a$10$F9ey7zW.sVliT224RFxpWeMsgzO.D9YRG54a8T36/K2MCiT41nzmC"
  ```

- **spec.sshPublicKey**
  Deprecated
  string

  Node user SSH public key.

  Either `sshPublicKey` or `sshPublicKeys` **must** be specified.

  Example:

  ```
  sshPublicKey: ssh-rsa AAABBB
  ```

- **spec.sshPublicKeys**
  array of strings

  Node user SSH public keys.

  Either `sshPublicKey` or `sshPublicKeys` **must** be specified.

  Example:

  ```
  sshPublicKeys:
  - ssh-rsa AAABBB
  - cert-authority,principals="name" ssh-rsa BBBCCC
  ```

- **spec.uid**
  number

  Required value

  Node user ID.

  We recommend using the values `>= 1100` to avoid conflicts with manually created users.

  This parameter does not change during the entire resource life.

  Allowed values: `1001 <= X`

  Example:

```
uid: 1100
```

- **status**
  object

  User creation status information

  - **status.errors**
    object

    List of user creation errors on each node

Deprecated resource. Support for the resource might be removed in a later release.

Defines the linux users to create on all nodes.

The user’s home directory is created in the `/home/deckhouse/` directory.

- **spec**
  object

  Required value

  - **spec.extraGroups**
    array of strings

    Node user additional system groups.

    Examples:

    ```
    extraGroups:
    - docker
    ```

    ```
    extraGroups:
    - docker
    - ftp
    ```

  - **spec.isSudoer**
    boolean

    Persistence of node user in sudo group.

    Default: `false`

    Example:

    ```
    isSudoer: true
    ```

  - **spec.nodeGroups**
    array of strings

    List of NodeGroups to apply the user for.

    Default: `["*"]`

    Examples:

    ```
    nodeGroups:
    - master
    - worker
    ```

    ```
    nodeGroups:
    - worker
    ```

    ```
    nodeGroups:
    - "*"
    ```

  - **spec.passwordHash**
    string

    Hashed user password.

    The format corresponds to the password hashes in `/etc/shadow`. You can get it using the following command: `openssl passwd -6`.

Example:

```
passwordHash: "$2a$10$F9ey7zW.sVliT224RFxpWeMsgzO.D9YRG54a8T36/K2MCiT41nzmC"
```

- **spec.sshPublicKey**
  Deprecated
  string

  Node user SSH public key.

  Either `sshPublicKey` or `sshPublicKeys` **must** be specified.

  Example:

  ```
  sshPublicKey: ssh-rsa AAABBB
  ```

- **spec.sshPublicKeys**
  array of strings

  Node user SSH public keys.

  Either `sshPublicKey` or `sshPublicKeys` **must** be specified.

  Example:

  ```
  sshPublicKeys:
  - ssh-rsa AAABBB
  - cert-authority,principals="name" ssh-rsa BBBCCC
  ```

- **spec.uid**
  number

  Required value

  Node user ID.

  We recommend using the values `>= 1100` to avoid conflicts with manually created users.

  This parameter does not change during the entire resource life.

  Allowed values: `1001 <= X`

  Example:

  ```
  uid: 1100
  ```

SSHCredentials

Scope: Cluster

- v1alpha2
- v1alpha1

Contains credentials required by Cluster API Provider Static (CAPS) to connect over SSH. CAPS connects to the server (virtual machine) defined in the StaticInstance custom resource to manage its state.

A reference to this resource is specified in the credentialsRef parameter of the `StaticInstance` resource.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

SSHCredentialsSpec defines the desired state of SSHCredentials.

- **spec.privateSSHKey**
  string

  Private SSH key in PEM format encoded as base64 string.

- **spec.sshExtraArgs**
  string

  A list of additional arguments to pass to the openssh command.

  Examples:

  ```
  sshExtraArgs: "-vvv"
  ```

  ```
  sshExtraArgs: "-c chacha20-poly1305@openssh.com"
  ```

  ```
  sshExtraArgs: "-c aes256-gcm@openssh.com"
  ```

  ```
  sshExtraArgs: "-m umac-64-etm@openssh.com"
  ```

  ```
  sshExtraArgs: "-m hmac-sha2-512-etm@openssh.com"
  ```

- **spec.sshPort**
  integer

  A port to connect to the host via SSH.

  Default: `22`

  Allowed values: `1 <= X <= 65535`

- **spec.sudoPasswordEncoded**
  string

  Base64 encoded sudo password for the user.

- **spec.user**
  string

  Required value

  A username to connect to the host via SSH.

Contains credentials required by Cluster API Provider Static (CAPS) to connect over SSH. CAPS connects to the server (virtual machine) defined in the StaticInstance custom resource to manage its state.

A reference to this resource is specified in the credentialsRef parameter of the `StaticInstance` resource.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

  SSHCredentialsSpec defines the desired state of SSHCredentials.

  - **spec.privateSSHKey**
    string

    Private SSH key in PEM format encoded as base64 string.

  - **spec.sshExtraArgs**

string

A list of additional arguments to pass to the openssh command.

Examples:

```
sshExtraArgs: "-vvv"
```

```
sshExtraArgs: "-c chacha20-poly1305@openssh.com"
```

```
sshExtraArgs: "-c aes256-gcm@openssh.com"
```

```
sshExtraArgs: "-m umac-64-etm@openssh.com"
```

```
sshExtraArgs: "-m hmac-sha2-512-etm@openssh.com"
```

- **spec.sshPort**
  integer

  A port to connect to the host via SSH.

  Default: `22`

  Allowed values: `1 <= X <= 65535`

- **spec.sudoPassword**
  string

  A sudo password for the user.

- **spec.user**
  string

  Required value

  A username to connect to the host via SSH.

StaticInstance

Scope: Cluster

- v1alpha2
- v1alpha1

StaticInstance describes a machine for the Cluster API Provider Static.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

  StaticInstanceSpec defines the desired state of StaticInstance.

  - **spec.address**
    string

    Required value

    The IP address of the host.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  - **spec.credentialsRef**

object

Required value

The reference to the `SSHCredentials` object.

- **spec.credentialsRef.apiVersion**
  string

  API version of the referent.

- **spec.credentialsRef.fieldPath**
  string

  If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as desiredState.manifest.containers[2]. For example, if the object reference is to a container within a pod, this would take on a value like: “spec.containers{name}” (where “name” refers to the name of the container that triggered the event) or if no container name is specified “spec.containers[2]” (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

- **spec.credentialsRef.kind**
  string

  Kind of the referent. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **spec.credentialsRef.name**
  string

  Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.credentialsRef.namespace**
  string

  Namespace of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

- **spec.credentialsRef.resourceVersion**
  string

  Specific resourceVersion to which this reference is made, if any. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency

- **spec.credentialsRef.uid**
  string

  UID of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids

- **status**
  object

  StaticInstanceStatus defines the observed state of StaticInstance.

  - **status.conditions**
    array of objects

    Conditions defines current service state of the StaticInstance.

    Condition contains details for one aspect of the current state of this API Resource.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

    - **status.conditions.message**
      string

      Required value

      message is a human readable message indicating details about the transition. This may be an empty string.

Maximum length: `32768`

- **status.conditions.observedGeneration**
  integer

  observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

  Allowed values: `0 <= X`

- **status.conditions.reason**
  string

  Required value

  reason contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

  Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

  Length: `1..1024`

- **status.conditions.status**
  string

  Required value

  status of the condition, one of True, False, Unknown.

  Allowed values: `True`, `False`, `Unknown`

- **status.conditions.type**
  string

  Required value

  type of condition in CamelCase or in foo.example.com/CamelCase.

  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

  Maximum length: `316`

- **status.currentStatus**
  object
  - **status.currentStatus.lastUpdateTime**
    string
  - **status.currentStatus.phase**
    string

    Allowed values: `Error`, `Pending`, `Bootstrapping`, `Running`, `Cleaning`

- **status.machineRef**
  object

  The reference to the `StaticMachine` object.

  - **status.machineRef.apiVersion**
    string

    API version of the referent.

  - **status.machineRef.fieldPath**
    string

    If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as desiredState.manifest.containers[2]. For example, if the object reference is to a container within a pod, this would take on a value like: "spec.containers{name}" (where "name" refers to the name of the container that triggered the event) or if no container name is specified "spec.containers[2]" (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

  - **status.machineRef.kind**
    string

    Kind of the referent. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

        - **status.machineRef.name**
          string

          Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

        - **status.machineRef.namespace**
          string

          Namespace of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

        - **status.machineRef.resourceVersion**
          string

          Specific resourceVersion to which this reference is made, if any. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency

        - **status.machineRef.uid**
          string

          UID of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids

    - **status.nodeRef**
      object

      The reference to the `Node` object.

        - **status.nodeRef.apiVersion**
          string

          API version of the referent.

        - **status.nodeRef.fieldPath**
          string

          If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as desiredState.manifest.containers[2]. For example, if the object reference is to a container within a pod, this would take on a value like: “spec.containers{name}” (where “name” refers to the name of the container that triggered the event) or if no container name is specified “spec.containers[2]” (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

        - **status.nodeRef.kind**
          string

          Kind of the referent. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

        - **status.nodeRef.name**
          string

          Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

        - **status.nodeRef.namespace**
          string

          Namespace of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

        - **status.nodeRef.resourceVersion**
          string

          Specific resourceVersion to which this reference is made, if any. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency

        - **status.nodeRef.uid**
          string

          UID of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids

StaticInstance describes a machine for the Cluster API Provider Static.

- **apiVersion**
  string

APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

  StaticInstanceSpec defines the desired state of StaticInstance.

  - **spec.address**
    string

    Required value

    The IP address of the host.

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$`

  - **spec.credentialsRef**
    object

    Required value

    The reference to the `SSHCredentials` object.

    - **spec.credentialsRef.apiVersion**
      string

      API version of the referent.

    - **spec.credentialsRef.fieldPath**
      string

      If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as desiredState.manifest.containers[2]. For example, if the object reference is to a container within a pod, this would take on a value like: “spec.containers{name}” (where “name” refers to the name of the container that triggered the event) or if no container name is specified “spec.containers[2]” (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

    - **spec.credentialsRef.kind**
      string

      Kind of the referent. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

    - **spec.credentialsRef.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.credentialsRef.namespace**
      string

      Namespace of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

    - **spec.credentialsRef.resourceVersion**
      string

      Specific resourceVersion to which this reference is made, if any. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency

    - **spec.credentialsRef.uid**
      string

      UID of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids

- **status**

object

StaticInstanceStatus defines the observed state of StaticInstance.

- **status.conditions**
  array of objects

  Conditions defines current service state of the StaticInstance.

  Condition contains details for one aspect of the current state of this API Resource.

  - **status.conditions.lastTransitionTime**
    string

    Required value

    lastTransitionTime is the last time the condition transitioned from one status to another. This should be when the underlying condition changed. If that is not known, then using the time when the API field changed is acceptable.

  - **status.conditions.message**
    string

    Required value

    message is a human readable message indicating details about the transition. This may be an empty string.

    Maximum length: `32768`

  - **status.conditions.observedGeneration**
    integer

    observedGeneration represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

    Allowed values: `0 <= X`

  - **status.conditions.reason**
    string

    Required value

    reason contains a programmatic identifier indicating the reason for the condition’s last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API. The value should be a CamelCase string. This field may not be empty.

    Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

    Length: `1..1024`

  - **status.conditions.status**
    string

    Required value

    status of the condition, one of True, False, Unknown.

    Allowed values: `True`, `False`, `Unknown`

  - **status.conditions.type**
    string

    Required value

    type of condition in CamelCase or in foo.example.com/CamelCase.

    Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

    Maximum length: `316`

- **status.currentStatus**
  object
  - **status.currentStatus.lastUpdateTime**
    string
  - **status.currentStatus.phase**
    string

    Allowed values: `Error`, `Pending`, `Bootstrapping`, `Running`, `Cleaning`

- **status.machineRef**
  object

  The reference to the `StaticMachine` object.

  - **status.machineRef.apiVersion**
    string

    API version of the referent.

  - **status.machineRef.fieldPath**
    string

    If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as desiredState.manifest.containers[2]. For example, if the object reference is to a container within a pod, this would take on a value like: “spec.containers{name}” (where “name” refers to the name of the container that triggered the event) or if no container name is specified “spec.containers[2]” (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

  - **status.machineRef.kind**
    string

    Kind of the referent. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

  - **status.machineRef.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **status.machineRef.namespace**
    string

    Namespace of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

  - **status.machineRef.resourceVersion**
    string

    Specific resourceVersion to which this reference is made, if any. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency

  - **status.machineRef.uid**
    string

    UID of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids

- **status.nodeRef**
  object

  The reference to the `Node` object.

  - **status.nodeRef.apiVersion**
    string

    API version of the referent.

  - **status.nodeRef.fieldPath**
    string

    If referring to a piece of an object instead of an entire object, this string should contain a valid JSON/Go field access statement, such as desiredState.manifest.containers[2]. For example, if the object reference is to a container within a pod, this would take on a value like: “spec.containers{name}” (where “name” refers to the name of the container that triggered the event) or if no container name is specified “spec.containers[2]” (container with index 2 in this pod). This syntax is chosen only to have some well-defined way of referencing a part of an object.

  - **status.nodeRef.kind**
    string

    Kind of the referent. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

  - **status.nodeRef.name**
    string

Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **status.nodeRef.namespace**
  string

  Namespace of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

- **status.nodeRef.resourceVersion**
  string

  Specific resourceVersion to which this reference is made, if any. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#concurrency-control-and-consistency

- **status.nodeRef.uid**
  string

  UID of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#uids

FAQ

> **i** The module lifecycle stage: General Availability

How do I add master nodes to a cloud cluster?

For the procedure of converting a cluster with a single master node to a multi-master cluster, refer to the FAQ of the `control-plane-manager` module.

How do I reduce the number of master nodes in a cloud cluster?

For the procedure of converting a multi-master cluster to a cluster with a single master node, refer to the FAQ of the `control-plane-manager` module.

Static nodes

You can add a static node to the cluster manually (an example) or by using Cluster API Provider Static.

How do I add a static node to a cluster (Cluster API Provider Static)?

To add a static node to a cluster (bare metal server or virtual machine), follow these steps:

1. Prepare the required resources:
   - Allocate a server or virtual machine and ensure that the node has the necessary network connectivity with the cluster.
   - If necessary, install additional operating system packages and configure the mount points that will be used on the node.
2. Create a user with `sudo` privileges:
   - Add a new user (in this example, `caps`) with `sudo` privileges:

     ```
     useradd -m -s /bin/bash caps
     usermod -aG sudo caps
     ```

   - Allow the user to run `sudo` commands without having to enter a password. For this, add the following line to the `sudo` configuration on the server (you can either edit the `/etc/sudoers` file, or run the `sudo visudo` command, or use some other method):

     ```
     caps ALL=(ALL) NOPASSWD: ALL
     ```

3. Set `UsePAM` to `yes` in `/etc/ssh/sshd_config` on server and restart sshd service:

   ```
   sudo systemctl restart sshd
   ```

4. Generate a pair of SSH keys with an empty passphrase on the server:

```
ssh-keygen -t ed25519 -f caps-id -C "" -N ""
```

The public and private keys of the caps user will be stored in the `caps-id.pub` and `caps-id` files in the current directory on the server.

5. Add the generated public key to the `/home/caps/.ssh/authorized_keys` file of the `caps` user by executing the following commands in the keys directory on the server:

```
mkdir -p /home/caps/.ssh
cat caps-id.pub >> /home/caps/.ssh/authorized_keys
chmod 700 /home/caps/.ssh
chmod 600 /home/caps/.ssh/authorized_keys
chown -R caps:caps /home/caps/
```

6. Create the SSHCredentials resource.
7. Create the StaticInstance resource.
8. Create the NodeGroup resource with the `Static` nodeType, specify the desired number of nodes in the group and, if necessary, the filter for `StaticInstance`.

An example of adding a static node.

How do I add a batch of static nodes to a cluster manually?

Use an existing one or create a new NodeGroup custom resource (example of the `NodeGroup` called `worker`). The nodeType parameter for static nodes in the NodeGroup must be `Static` or `CloudStatic`.

You can automate the bootstrap process with any automation platform you prefer. The following is an example for Ansible.

1. Pick up one of Kubernetes API Server endpoints. Note that this IP must be accessible from nodes that are being added to the cluster:

```
d8 k -n default get ep kubernetes -o json | jq '.subsets[0].addresses[0].ip + ":" +
(.subsets[0].ports[0].port | tostring)' -r
```

Check the K8s version. If the version >= 1.25, create `node-group` token:

```
d8 k create token node-group --namespace d8-cloud-instance-manager --duration 1h
```

Save the token you got and add it to the `token:` field of the Ansible playbook in the next steps.

2. If the Kubernetes version is smaller than 1.25, get a Kubernetes API token for a special ServiceAccount that Deckhouse manages:

```
d8 k -n d8-cloud-instance-manager get $(d8 k -n d8-cloud-instance-manager get secret -o name | grep
node-group-token) \
  -o json | jq '.data.token' -r | base64 -d && echo ""
```

3. Create Ansible playbook with `vars` replaced with values from previous steps:

```
- hosts: all
  become: yes
  gather_facts: no
  vars:
    kube_apiserver: <KUBE_APISERVER>
    token: <TOKEN>
  tasks:
    - name: Check if node is already bootsrapped
      stat:
        path: /var/lib/bashible
      register: bootstrapped
    - name: Get bootstrap secret
      uri:
        url: "https://{{ kube_apiserver }}/api/v1/namespaces/d8-cloud-instance-
manager/secrets/manual-bootstrap-for-{{ node_group }}"
        return_content: yes
        method: GET
        status_code: 200
        body_format: json
        headers:
          Authorization: "Bearer {{ token }}"
        validate_certs: no
      register: bootstrap_secret
      when: bootstrapped.stat.exists == False
    - name: Run bootstrap.sh
      shell: "{{ bootstrap_secret.json.data['bootstrap.sh'] | b64decode }}"
      args:
        executable: /bin/bash
      ignore_errors: yes
      when: bootstrapped.stat.exists == False
    - name: wait
      wait_for_connection:
        delay: 30
      when: bootstrapped.stat.exists == False
```

4. Specify one more `node_group` variable. This variable must be the same as the name of `NodeGroup` to which node will belong. Variable can be passed in different ways, for example, by using an inventory file.:

```
[system]
system-0
system-1

[system:vars]
node_group=system

[worker]
worker-0
worker-1

[worker:vars]
node_group=worker
```

5. Run the playbook with the inventory file.

How do I clean up a static node manually?

> **i** This method is valid for both manually configured nodes (using the bootstrap script) and nodes configured using CAPS.

To decommission a node from the cluster and clean up the server (VM), run the following command on the node:

For all nodes except the control-plane

1. Delete a node from the Kubernetes cluster:

```
d8 k drain <node>
d8 k drain <node> --ignore-daemonsets --delete-emptydir-data
d8 k delete pods --all-namespaces --field-selector spec.nodeName=<node> --force
d8 k delete node <node>
```

2. Run the cleanup script on the node:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
```

3. After restarting the node run the script `bootstrap.sh`.

For control-plane nodes

1. Remove the labels `node-role.kubernetes.io/control-plane`, `node-role.kubernetes.io/master`, and `node.deckhouse.io/group` from the node:

```
d8 k label node <node> node-role.kubernetes.io/control-plane- node-role.kubernetes.io/master- node.deckhouse.io/group-
```

2. Make sure the removed node with control-plane has disappeared from the list of etcd cluster members:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | head -n1) -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key --endpoints https://127.0.0.1:2379/ member list -w table
```

3. Delete a node from the Kubernetes cluster:

```
d8 k drain <node>
d8 k drain <node> --ignore-daemonsets --delete-emptydir-data
d8 k delete pods --all-namespaces --field-selector spec.nodeName=<node> --force
d8 k delete node <node>
```

4. Run the cleanup script on the node:

```
bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
```

5. After restarting the node run the script `bootstrap.sh`.
6. Wait for the Deckhouse queues to be processed and ensure that the etcd cluster member has reappeared in the list:

```
d8 k -n kube-system exec -ti $(d8 k -n kube-system get pod -l component=etcd,tier=control-plane -o name | head -n1) -- etcdctl --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key --endpoints https://127.0.0.1:2379/ member list -w table
```

Can I delete a StaticInstance?

A StaticInstance that is in the `Pending` state can be deleted with no adverse effects.

To delete a StaticInstance in any state other than `Pending` (`Running`, `Cleaning`, `Bootstrapping`), you need to:

1. Add the label `"node.deckhouse.io/allow-bootstrap": "false"` to the StaticInstance.

   Example command for adding a label:

```
d8 k label staticinstance d8cluster-worker node.deckhouse.io/allow-bootstrap=false
```

2. Wait until the StaticInstance status becomes `Pending`.

   To check the status of StaticInstance, use the command:

```
d8 k get staticinstances
```

3. Delete the StaticInstance.

   Example command for deleting StaticInstance:

   ```
   d8 k delete staticinstance d8cluster-worker
   ```

4. Decrease the `NodeGroup.spec.staticInstances.count` field by 1.

How do I change the IP address of a StaticInstance?

You cannot change the IP address in the `StaticInstance` resource. If an incorrect address is specified in `StaticInstance`, you have to delete the StaticInstance and create a new one.

How do I migrate a manually configured static node under CAPS control?

You need to clean up the node, then hand over the node under CAPS control.

How do I change the NodeGroup of a static node?

Note that if a node is under CAPS control, you **cannot** change the `NodeGroup` membership of such a node. The only alternative is to delete StaticInstance and create a new one.

To switch an existing manually created static node to another `NodeGroup`, you need to change its group label:

```
d8 k label node --overwrite <node_name> node.deckhouse.io/group=<new_node_group_name>
d8 k label node <node_name> node-role.kubernetes.io/<old_node_group_name>-
```

Applying the changes will take some time.

How to clean up a node for adding to the cluster?

This is only needed if you have to move a static node from one cluster to another. Be aware these operations remove local storage data. If you just need to change a NodeGroup, follow this instruction.

> ⚠ Evict resources from the node and remove the node from LINSTOR/DRBD using the instruction if the node you are cleaning up has LINSTOR/DRBD storage pools.

1. Delete the node from the Kubernetes cluster:

   ```
   d8 k drain <node> --ignore-daemonsets --delete-emptydir-data
   d8 k delete node <node>
   ```

2. Run cleanup script on the node:

   ```
   bash /var/lib/bashible/cleanup_static_node.sh --yes-i-am-sane-and-i-understand-what-i-am-doing
   ```

3. Run the `bootstrap.sh` script after reboot of the node.

How do I know if something went wrong?

If a node in a nodeGroup is not updated (the value of `UPTODATE` when executing the `d8 k get nodegroup` command is less than the value of `NODES`) or you assume some other problems that may be related to the `node-manager` module, then you need to look at the logs of the `bashible` service. The `bashible` service runs on each node managed by the `node-manager` module.

To view the logs of the `bashible` service on a specific node, run the following command:

```
journalctl -fu bashible
```

Example of output when the `bashible` service has performed all necessary actions:

```
May 25 04:39:16 kube-master-0 systemd[1]: Started Bashible service.
May 25 04:39:16 kube-master-0 bashible.sh[1976339]: Configuration is in sync, nothing to
do.
May 25 04:39:16 kube-master-0 systemd[1]: bashible.service: Succeeded.
```

How do I know what is running on a node while it is being created?

You can analyze `cloud-init` to find out what's happening on a node during the bootstrapping process:

1. Find the node that is currently bootstrapping:

   ```
   d8 k get instances | grep Pending
   ```

   An example:

   ```
   d8 k get instances | grep Pending
   dev-worker-2a6158ff-6764d-nrtbj   Pending   46s
   ```

2. Get information about connection parameters for viewing logs:

   ```
   d8 k get instances dev-worker-2a6158ff-6764d-nrtbj -o yaml | grep 'bootstrapStatus' -B0 -A2
   ```

   An example:

   ```
   d8 k get instances dev-worker-2a6158ff-6764d-nrtbj -o yaml | grep 'bootstrapStatus' -B0 -A2
   bootstrapStatus:
     description: Use 'curl -N http://192.168.199.158:8000' to get bootstrap logs.
     logsEndpoint: http://192.168.199.158:8000
   ```

3. Run the command you got (`curl -N http://192.168.199.158:8000` according to the example above) to see `cloud-init` logs and determine the cause of the problem on the node.

The logs of the initial node configuration are located at `/var/log/cloud-init-output.log`.

How do I update kernel on nodes?

Debian-based distros

Create a `Node Group Configuration` resource by specifying the desired kernel version in the `desired_version` variable of the shell script (the resource's spec.content parameter):

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-kernel.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 32
  content: |
    # Copyright 2022 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    desired_version="5.15.0-53-generic"

    bb-event-on 'bb-package-installed' 'post-install'
    post-install() {
      bb-log-info "Setting reboot flag due to kernel was updated"
      bb-flag-set reboot
    }

    version_in_use="$(uname -r)"

    if [[ "$version_in_use" == "$desired_version" ]]; then
      exit 0
    fi

    bb-deckhouse-get-disruptive-update-approval
    bb-apt-install "linux-image-${desired_version}"
```

CentOS-based distros

Create a `Node Group Configuration` resource by specifying the desired kernel version in the `desired_version` variable of the shell script (the resource’s spec.content parameter):

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: install-kernel.sh
spec:
  bundles:
    - '*'
  nodeGroups:
    - '*'
  weight: 32
  content: |
    # Copyright 2022 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    desired_version="3.10.0-1160.42.2.el7.x86_64"

    bb-event-on 'bb-package-installed' 'post-install'
    post-install() {
      bb-log-info "Setting reboot flag due to kernel was updated"
      bb-flag-set reboot
    }

    version_in_use="$(uname -r)"

    if [[ "$version_in_use" == "$desired_version" ]]; then
      exit 0
    fi

    bb-deckhouse-get-disruptive-update-approval
    bb-dnf-install "kernel-${desired_version}"
```

NodeGroup parameters and their result

The NodeGroup parameter	Disruption update	Node provisioning	Kubelet restart
chaos	-	-	-
cloudInstances.classReference	-	+	-
cloudInstances.maxSurgePerZone	-	-	-
cri.containerd.maxConcurrentDownloads	-	-	+
cri.type	- (NotManaged) / + (other)	-	-
disruptions	-	-	-
kubelet.maxPods	-	-	+
kubelet.rootDir	-	-	+
kubernetesVersion	-	-	+
nodeTemplate	-	-	-

The NodeGroup parameter	Disruption update	Node provisioning	Kubelet restart
static	-	-	+
update.maxConcurrent	-	-	-

Refer to the description of the NodeGroup custom resource for more information about the parameters.

When the `InstanceClass` or `instancePrefix` parameters are modified, the process is similar to updating a Deployment in Kubernetes (changing `InstanceClass` results in creating a new MachineSet, and changing `instancePrefix` results in creating a new MachineDeployment). New instances will be created according to the value of the `cloudInstances.maxSurgePerZone` parameter. As the new instances are created, the old ones will be gradually removed, with a drain operation performed beforehand.

During the disruption update, an evict of the pods from the node is performed. If any pod fails to evict, the evict is repeated every 20 seconds until a global timeout of 5 minutes is reached. After that, the pods that failed to evict are removed.

How do I redeploy ephemeral machines in the cloud with a new configuration?

If the Deckhouse configuration is changed (both in the node-manager module and in any of the cloud providers), the VMs will not be redeployed. The redeployment is performed only in response to changing `InstanceClass` or `NodeGroup` objects.

To force the redeployment of all Machines, you need to add/modify the `manual-rollout-id` annotation to the `NodeGroup`: `d8 k annotate NodeGroup name_ng "manual-rollout-id=$(uuidgen)" --overwrite`.

How do I allocate nodes to specific loads?

⚠ You cannot use the `deckhouse.io` domain in `labels` and `taints` keys of the `NodeGroup`. It is reserved for Deckhouse components. Please, use the `dedicated` or `dedicated.client.com` keys.

There are two ways to solve this problem:

1. You can set labels to `NodeGroup`'s `spec.nodeTemplate.labels`, to use them in the `Pod`'s spec.nodeSelector or spec.affinity.nodeAffinity parameters. In this case, you select nodes that the scheduler will use for running the target application.
2. You cat set taints to `NodeGroup`'s `spec.nodeTemplate.taints` and then remove them via the `Pod`'s spec.tolerations parameter. In this case, you disallow running applications on these nodes unless those applications are explicitly allowed.

i Deckhouse tolerates the `dedicated` by default, so we recommend using the `dedicated` key with any `value` for taints on your dedicated nodes.

To use custom keys for `taints` (e.g., `dedicated.client.com`), you must add the key's value to the array `.spec.settings.modules.placement.customTolerationKeys` parameters. This way, deckhouse can deploy system components (e.g., `cni-flannel`) to these dedicated nodes.

How to allocate nodes to system components?

Frontend

For Ingress controllers, use the `NodeGroup` with the following configuration:

```
nodeTemplate:
  labels:
    node-role.deckhouse.io/frontend: ""
  taints:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: frontend
```

System components

`NodeGroup` for components of Deckhouse subsystems will look as follows:

```
nodeTemplate:
  labels:
    node-role.deckhouse.io/system: ""
  taints:
    - effect: NoExecute
      key: dedicated.deckhouse.io
      value: system
```

How do I speed up node provisioning on the cloud when scaling applications horizontally?

The most efficient way is to have some extra nodes “ready”. In this case, you can run new application replicas on them almost instantaneously. The obvious disadvantage of this approach is the additional maintenance costs related to these nodes.

Here is how you should configure the target `NodeGroup`:

1. Specify the number of “ready” nodes (or a percentage of the maximum number of nodes in the group) using the `cloudInstances.standby` parameter.
2. If there are additional service components on nodes that are not handled by Deckhouse (e.g., the `filebeat` DaemonSet), you can specify the percentage of node resources they can consume via the `standbyHolder.overprovisioningRate` parameter.
3. This feature requires that at least one group node is already running in the cluster. In other words, there must be either a single replica of the application, or the `cloudInstances.minPerZone` parameter must be set to `1`.

An example:

```
cloudInstances:
  maxPerZone: 10
  minPerZone: 1
  standby: 10%
  standbyHolder:
    overprovisioningRate: 30%
```

How do I disable machine-controller-manager/CAPI in the case of potentially cluster-damaging changes?

✖ Use this switch only if you know what you are doing and clearly understand the consequences.

Set the `mcmEmergencyBrake` parameter to true:

```
mcmEmergencyBrake: true
```

For disabling CAPI, set the `capiEmergencyBrake` parameter to true:

```
capiEmergencyBrake: true
```

How do I restore the master node if kubelet cannot load the control plane components?

Such a situation may occur if images of the control plane components on the master were deleted in a cluster that has a single master node (e.g., the directory `/var/lib/containerd` was deleted). In this case, kubelet cannot pull images of the control plane components when restarted since the master node lacks authorization parameters required for accessing `registry.deckhouse.io`.

Below is an instruction on how you can restore the master node.

containerd

Execute the following command to restore the master node in any cluster running under Deckhouse:

```
d8 k -n d8-system get secrets deckhouse-registry -o json |
jq -r '.data.".dockerconfigjson"' | base64 -d |
jq -r '.auths."registry.deckhouse.io".auth'
```

Copy the command’s output and use it for setting the `AUTH` variable on the corrupted master.

Next, you need to pull images of `control plane` components to the corrupted master:

```
for image in $(grep "image:" /etc/kubernetes/manifests/* | awk '{print $3}'); do
  crictl pull --auth $AUTH $image
done
```

You need to restart `kubelet` after pulling the images.

How to change CRI for NodeGroup?

⚠ CRI can only be switched from `Containerd` to `NotManaged` and back (the cri.type parameter).

Set NodeGroup `cri.type` to `Containerd` or `NotManaged`.

NodeGroup YAML example:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  cri:
    type: Containerd
```

Also, this operation can be done with patch:

- For `Containerd`:

  ```
  d8 k patch nodegroup <NodeGroup name> --type merge -p '{"spec":{"cri":{"type":"Containerd"}}}'
  ```

- For `NotManaged`:

  ```
  d8 k patch nodegroup <NodeGroup name> --type merge -p '{"spec":{"cri":{"type":"NotManaged"}}}'
  ```

⚠ While changing `cri.type` for NodeGroups, created using `dhctl`, you must change it in `dhctl config edit provider-cluster-configuration` and in `NodeGroup` object.

After setting up a new CRI for NodeGroup, the node-manager module drains nodes one by one and installs a new CRI on them. Node update is accompanied by downtime (disruption). Depending on the `disruption` setting for NodeGroup, the node-manager module either automatically allows node updates or requires manual confirmation.

Why might the CRI change not apply?

When attempting to switch the CRI, the changes may not take effect. The most common reason is the presence of special node labels: `node.deckhouse.io/containerd-v2-unsupported` and `node.deckhouse.io/containerd-config=custom`.

The `node.deckhouse.io/containerd-v2-unsupported` label is set to a node if at least one of the following conditions is true:

- Kernel version lower than 5.8
- Linux kernel version is in the range 6.12.0–6.12.28 or 6.14.0–6.14.6 (these versions are affected by CVE-2025-37999 in EROFS)
- systemd version lower than 244
- cgroup v2 is disabled
- EROFS file system is unavailable.

The `node.deckhouse.io/containerd-config=custom` label is set if the node contains `.toml` files in the `/etc/containerd/conf.d/` (if CRI containerd v1 is used on the cluster nodes) or `/etc/containerd/conf2.d/` (if CRI containerd v2 is used on the cluster nodes) directories. In this case, you should remove such files (provided this will not have critical impact on running containers) and delete the corresponding NodeGroupConfiguration (NGC) resources through which they may have been added.

If the Deckhouse Virtualization Platform is used, an additional reason why the CRI may fail to switch can be the `containerd-dvcr-config.sh` NGC. If the virtualization platform is already installed and running, this

NGC can be removed.

If you cannot remove the NodeGroupConfiguration resource that modifies the containerd configuration and is incompatible with containerd v2, use the universal template:

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
spec:
  bundles:
  - '*'
  content: |
    {{- if eq .cri "ContainerdV2" }}
  # <Script to modify the configuration for ContainerdV2>.
    {{- else }}
  # <Script to modify the configuration for ContainerdV1>.
    {{- end }}
  nodeGroups:
  - '*'
  weight: 31
```

Additionally, to switch the CRI you may need to remove the custom label `node.deckhouse.io/containerd-config=custom`. You can do this with the following command:

```
for node in $(d8 k get nodes -l node-role.kubernetes.io/<Name of NodeGroup where CRI is changed>=); do kubectl label $node node.deckhouse.io/containerd-config-; done
```

How to change CRI for the whole cluster?

⚠ CRI can only be switched from `Containerd` to `NotManaged` and back (the cri.type parameter).

It is necessary to use the `dhctl` utility to edit the `defaultCRI` parameter in the `cluster-configuration` config.

Also, this operation can be done with the following patch:

- For `Containerd`:

  ```
  data="$(d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r '.data."cluster-configuration.yaml"' | base64 -d | sed "s/NotManaged/Containerd/" | base64 -w0)"
  d8 k -n kube-system patch secret d8-cluster-configuration -p "{\"data\":{\"cluster-configuration.yaml\":\"$data\"}}"
  ```

- For `NotManaged`:

  ```
  data="$(d8 k -n kube-system get secret d8-cluster-configuration -o json | jq -r '.data."cluster-configuration.yaml"' | base64 -d | sed "s/Containerd/NotManaged/" | base64 -w0)"
  d8 k -n kube-system patch secret d8-cluster-configuration -p "{\"data\":{\"cluster-configuration.yaml\":\"$data\"}}"
  ```

If it is necessary to leave some NodeGroup on another CRI, then before changing the `defaultCRI` it is necessary to set CRI for this NodeGroup, as described here.

✖ Changing `defaultCRI` entails changing CRI on all nodes, including master nodes. If there is only one master node, this operation is dangerous and can lead to a complete failure of the cluster! The preferred option is to make a multi-master and change the CRI type.

When changing the CRI in the cluster, additional steps are required for the master nodes:

1. Deckhouse updates nodes in master NodeGroup one by one, so you need to discover which node is updating right now:

```
d8 k get nodes -l node-role.kubernetes.io/control-plane="" -o json | jq '.items[] |
select(.metadata.annotations."update.node.deckhouse.io/approved"=="") |
.metadata.name' -r
```

2. Confirm the disruption of the master node that was discovered in the previous step:

   ```
   d8 k annotate node <master node name> update.node.deckhouse.io/disruption-approved=
   ```

3. Wait for the updated master node to switch to `Ready` state. Repeat steps for the next master node.

How to add node configuration step?

Additional node configuration steps are set via the NodeGroupConfiguration custom resource.

How to automatically put custom labels on the node?

1. On the node, create the directory `/var/lib/node_labels`.
2. Create a file or files containing the necessary labels in it. The number of files can be any, as well as the number of subdirectories containing them.
3. Add the necessary labels to the files in the `key=value` format. For example:

   ```
   example-label=test
   ```

4. Save the files.

When adding a node to the cluster, the labels specified in the files will be automatically affixed to the node.

⚠ Please note that it is not possible to add labels used in DKP in this way. This method will only work with custom labels that do not overlap with those reserved for Deckhouse.

How to deploy custom containerd configuration?

i The example of `NodeGroupConfiguration` uses functions of the script 032_configure_containerd.sh.

✖ Adding custom settings causes a restart of the containerd service.

Bashible on nodes merges main DKP containerd configuration with the following configuration files:

- `/etc/containerd/conf.d/*.toml`: If containerd v1 is used as the CRI on the cluster nodes.
- `/etc/containerd/conf2.d/*.toml`: If containerd v2 is used as the CRI on the cluster nodes.

⚠ You can override the values of the parameters that are specified in the file `/etc/containerd/deckhouse.toml`, but you will have to ensure their functionality on your own. Also, it is better not to change the configuration for the master nodes (nodeGroup `master`).

The following are configuration examples of the NodeGroupConfiguration resources adding a custom configuration file for a corresponding containerd version.

- For containerd v1
- For containerd v2

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-option-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2024 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_option.toml - << EOF
    oom_score = 500
    [metrics]
    address = "127.0.0.1"
    grpc_histogram = true
    EOF
  nodeGroups:
    - "worker"
  weight: 31
```

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: containerd-option-config.sh
spec:
  bundles:
    - '*'
  content: |
    # Copyright 2024 Flant JSC
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.

    mkdir -p /etc/containerd/conf2.d
    bb-sync-file /etc/containerd/conf2.d/runtimeclass.toml - << "EOF"
    [plugins."io.containerd.cri.v1.runtime".containerd.runtimes.gpu-large-shm]
      runtime_type = "io.containerd.runc.v2"
      [plugins."io.containerd.cri.v1.runtime".containerd.runtimes.gpu-large-shm.options]
        BinaryName = ""
        SystemdCgroup = true
        ShmSize = 17179869184
    EOF
  nodeGroups:
    - "worker"
  weight: 31
```

How to add configuration for an additional registry?

Containerd supports two methods for registry configuration: the **deprecated** method and the **actual** method.

To check for the presence of the **deprecated** configuration method, run the following commands on the cluster nodes:

```
cat /etc/containerd/config.toml | grep 'plugins."io.containerd.grpc.v1.cri".registry.mirrors'
cat /etc/containerd/config.toml | grep 'plugins."io.containerd.grpc.v1.cri".registry.configs'

# Example output:
# [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
#   [plugins."io.containerd.grpc.v1.cri".registry.mirrors."<REGISTRY_URL>"]
# [plugins."io.containerd.grpc.v1.cri".registry.configs]
#   [plugins."io.containerd.grpc.v1.cri".registry.configs."<REGISTRY_URL>".auth]
```

To check for the presence of the **actual** configuration method, run the following command on the cluster nodes:

```
cat /etc/containerd/config.toml | grep '/etc/containerd/registry.d'

# Example output:
# config_path = "/etc/containerd/registry.d"
```

Old Method

⚠ This containerd configuration format is deprecated.

i Used in containerd v1 when Deckhouse is not managed by the Registry module.

The configuration is described in the main containerd configuration file `/etc/containerd/config.toml`.

Adding custom configuration is carried out through the `toml merge` mechanism. Configuration files from the `/etc/containerd/conf.d` directory are merged with the main file `/etc/containerd/config.toml`. The merge takes place during the execution of the `032_configure_containerd.sh` script, so the corresponding files must be added in advance.

Example configuration file for the `/etc/containerd/conf.d/` directory:

```
[plugins]
  [plugins."io.containerd.grpc.v1.cri"]
    [plugins."io.containerd.grpc.v1.cri".registry]
      [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
          endpoint = ["https://${REGISTRY_URL}"]
      [plugins."io.containerd.grpc.v1.cri".registry.configs]
        [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
          auth = "${BASE_64_AUTH}"
          username = "${USERNAME}"
          password = "${PASSWORD}"
        [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
          ca_file = "${CERT_DIR}/${CERT_NAME}.crt"
          insecure_skip_verify = true
```

✖ Adding custom settings through the `toml merge` mechanism causes the containerd service to restart.

How to add additional registry auth (deprecated method)?

Example of adding authorization to a additional registry when using the **deprecated** configuration method:

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

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
              endpoint = ["https://${REGISTRY_URL}"]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]
            [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
              username = "username"
              password = "password"
              # OR
              auth = "dXNlcm5hbWU6cGFzc3dvcmQ="
    EOF
```

How to configure a certificate for an additional registry (deprecated method)?

Example of configuring a certificate for an additional registry when using the **deprecated** configuration method:

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

    REGISTRY_URL=private.registry.example
    CERT_FILE_NAME=${REGISTRY_URL}
    CERTS_FOLDER="/var/lib/containerd/certs/"


    mkdir -p ${CERTS_FOLDER}
    bb-sync-file "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" - << EOF
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    EOF

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]
            [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
              ca_file = "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"
    EOF
```

i In addition to containerd, the certificate can be added into the OS.

How to add TLS skip verify (deprecated method)?

Example of adding TLS skip verify when using the **deprecated** configuration method:

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

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]
            [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".tls]
              insecure_skip_verify = true
    EOF
```

After applying the configuration file, verify access to the registry from the nodes using the command:

```
# Via the CRI interface
crictl pull private.registry.example/image/repo:tag
```

How to set up a mirror for public image registries (deprecated method)?

Example of configuring a mirror for public image registries when using the **deprecated** configuration method:

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

    sed -i '/endpoint = \["https:\/\/registry-1.docker.io"\]/d'
/var/lib/bashible/bundle_steps/032_configure_containerd.sh
    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/mirror-to-harbor.toml - << "EOF"
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
              endpoint = ["https://registry.private.network/v2/dockerhub-proxy/"]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."gcr.io"]
              endpoint = ["https://registry.private.network/v2/YOUR_GCR_PROXY_REPO/"]
    EOF
```

New Method

> **i** Used in containerd v2.
>
> Used in containerd v1 when managed through the `registry` module (for example, in `Direct` mode).

The configuration is defined in the `/etc/containerd/registry.d` directory.
Configuration is specified by creating subdirectories named after the registry address:

```
/etc/containerd/registry.d
├── private.registry.example:5001
│   ├── ca.crt
│   └── hosts.toml
└── registry.deckhouse.io
    ├── ca.crt
    └── hosts.toml
```

Example contents of the `hosts.toml` file:

```
[host]
  # Mirror 1.
  [host."https://${REGISTRY_URL_1}"]
    capabilities = ["pull", "resolve"]
    ca = ["${CERT_DIR}/${CERT_NAME}.crt"]

    [host."https://${REGISTRY_URL_1}".auth]
      username = "${USERNAME}"
      password = "${PASSWORD}"

  # Mirror 2.
  [host."http://${REGISTRY_URL_2}"]
    capabilities = ["pull", "resolve"]
    skip_verify = true
```

i Configuration changes do not cause the containerd service to restart.

How to add additional registry auth (actual method)?

Example of adding authorization to a additional registry when using the **actual** configuration method:

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

    REGISTRY_URL=private.registry.example

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

How to configure a certificate for an additional registry (actual method)?

Example of configuring a certificate for an additional registry when using the **actual** configuration method:

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

    REGISTRY_URL=private.registry.example

    mkdir -p "/etc/containerd/registry.d/${REGISTRY_URL}"

    bb-sync-file "/etc/containerd/registry.d/${REGISTRY_URL}/ca.crt" - << EOF
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
    EOF

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

i In addition to containerd, the certificate can be added into the OS.

How to add TLS skip verify (actual method)?

Example of adding TLS skip verify when using the **actual** configuration method:

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

    REGISTRY_URL=private.registry.example

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

After applying the configuration file, check access to the registry from the nodes using the following commands:

```
# Via the CRI interface.
crictl pull private.registry.example/image/repo:tag

# Via ctr with the configuration directory specified.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/ private.registry.example/image/repo:tag

# Via ctr for an HTTP registry.
ctr -n k8s.io images pull --hosts-dir=/etc/containerd/registry.d/ --plain-http
private.registry.example/image/repo:tag
```

How to set up a mirror for public image registries (actual method)?

Example of configuring a mirror for public image registries when using the **actual** configuration method:

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

    REGISTRY1_URL=docker.io
    mkdir -p "/etc/containerd/registry.d/${REGISTRY1_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY1_URL}/hosts.toml" - << EOF
    [host."https://registry.private.network/v2/dockerhub-proxy/"]
      capabilities = ["pull", "resolve"]
      override_path = true
    EOF
    REGISTRY2_URL=gcr.io
    mkdir -p "/etc/containerd/registry.d/${REGISTRY2_URL}"
    bb-sync-file "/etc/containerd/registry.d/${REGISTRY2_URL}/hosts.toml" - << EOF
    [host."https://registry.private.network/v2/dockerhub-proxy/"]
      capabilities = ["pull", "resolve"]
      override_path = true
    EOF
```

How to use NodeGroup’s priority feature

The priority field of the `NodeGroup` CustomResource allows you to define the order in which nodes will be provisioned in the cluster. For example, `cluster-autoscaler` can first provision *spot-nodes* and switch to regular ones when they run out. Or it can provision larger nodes when there are plenty of resources in the cluster and then switch to smaller nodes once cluster resources run out.

Here is an example of creating two `NodeGroups` using spot-node nodes:

```
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker-spot
spec:
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker-spot
    maxPerZone: 5
    minPerZone: 0
    priority: 50
  nodeType: CloudEphemeral
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  cloudInstances:
    classReference:
      kind: AWSInstanceClass
      name: worker
    maxPerZone: 5
    minPerZone: 0
    priority: 30
  nodeType: CloudEphemeral
```

In the above example, `cluster-autoscaler` will first try to provision a spot-node. If it fails to add such a node to the cluster within 15 minutes, the `worker-spot` NodeGroup will be paused (for 20 minutes), and `cluster-autoscaler` will start provisioning nodes from the `worker` NodeGroup. If, after 30 minutes, another node needs to be deployed in the cluster, `cluster-autoscaler` will first attempt to provision a node from the `worker-spot` NodeGroup before provisioning one from the `worker` NodeGroup.

Once the `worker-spot` NodeGroup reaches its maximum (5 nodes in the example above), the nodes will be provisioned from the `worker` NodeGroup.

Note that node templates (labels/taints) for `worker` and `worker-spot` NodeGroups must be the same (or at least suitable for the load that triggers the cluster scaling process).

How to interpret Node Group states?

Ready — the node group contains the minimum required number of scheduled nodes with the status `Ready` for all zones.

Example 1. A group of nodes in the `Ready` state:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 1
status:
  conditions:
  - status: "True"
    type: Ready
---
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    node.deckhouse.io/group: ng1
status:
  conditions:
  - status: "True"
    type: Ready
```

Example 2. A group of nodes in the `Not Ready` state:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 2
status:
  conditions:
  - status: "False"
    type: Ready
---
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    node.deckhouse.io/group: ng1
status:
  conditions:
  - status: "True"
    type: Ready
```

Updating — a node group contains at least one node in which there is an annotation with the prefix `update.node.deckhouse.io` (for example, `update.node.deckhouse.io/waiting-for-approval`).

WaitingForDisruptiveApproval - a node group contains at least one node that has an annotation `update.node.deckhouse.io/disruption-required` and there is no annotation `update.node.deckhouse.io/disruption-approved`.

Scaling — calculated only for node groups with the type `CloudEphemeral`. The state `True` can be in two cases:

1. When the number of nodes is less than the *desired number of nodes* in the group, i.e. when it is necessary to increase the number of nodes in the group.
2. When a node is marked for deletion or the number of nodes is greater than the *desired number of nodes*, i.e. when it is necessary to reduce the number of nodes in the group.

The *desired number of nodes* is the sum of all replicas in the node group.

Example. The desired number of nodes is 2:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: ng1
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    maxPerZone: 5
    minPerZone: 2
status:
...
  desired: 2
...
```

Error — contains the last error that occurred when creating a node in a node group.

How do I make werf ignore the Ready conditions in a node group?

werf checks the `Ready` status of resources and, if available, waits for the value to become `True`.

Creating (updating) a nodeGroup resource in a cluster can take a significant amount of time to create the required number of nodes. When deploying such a resource in a cluster using werf (e.g., as part of a CI/CD process), deployment may terminate when resource readiness timeout is exceeded. To make werf ignore the nodeGroup status, the following `nodeGroup` annotations must be added:

```
metadata:
  annotations:
    werf.io/fail-mode: IgnoreAndContinueDeployProcess
    werf.io/track-termination-mode: NonBlocking
```

What is an Instance resource?

An Instance resource contains a description of an implementation-independent ephemeral machine resource. For example, machines created by MachineControllerManager or Cluster API Provider Static will have a corresponding Instance resource.

The object does not contain a specification. The status contains:

1. A link to the InstanceClass if it exists for this implementation;
2. A link to the Kubernetes Node object;
3. Current machine status;
4. Information on how to view machine creation logs (at the machine creation stage).

When a machine is created/deleted, the Instance object is created/deleted accordingly. You cannot create an Instance resource yourself, but you can delete it. In this case, the machine will be removed from the cluster (the removal process depends on implementation details).

When is a node reboot required?

Node reboots may be required after configuration changes. For example, after changing certain sysctl settings, specifically when modifying the `kernel.yama.ptrace_scope` parameter (e.g., using `astra-ptrace-lock enable/disable` in the Astra Linux distribution).

How to enable a delay before a node shutdown or restart while critical pods are running on it?

> ⚠ To decide whether to block a node shutdown, DKP additionally queries the NodeGroup. If the current node belongs to the `master` group and it is the only master node in the cluster, the shutdown block will not be applied to it.

To enable the mechanism that delays a pod's restart or shutdown, add the label `pod.deckhouse.io/inhibit-node-shutdown` to the Pod (for a Deployment, specify the label in the pod template).

Example of a Pod manifest with the `pod.deckhouse.io/inhibit-node-shutdown` label:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-stateful-app
  labels:
    app: my-stateful-app
    pod.deckhouse.io/inhibit-node-shutdown: "true"
spec:
  containers:
    - name: app
      image: my-registry/my-app:1.0.0
```

⚠ The block only applies to pods labeled with `pod.deckhouse.io/inhibit-node-shutdown` that are in the `Running` phase on the node. Kubelet shuts down the pods without this label according to the standard graceful shutdown process.

Searching pods that prevent shutdown and performing the node troubleshooting

To search the pods that are preventing a node shutdown, use the following command:

```
d8 k get po -A -l pod.deckhouse.io/inhibit-node-shutdown -o wide
```

To view the `GracefulShutdownPostpone` condition on a node, use the following command:

```
d8 k get node <NODE_NAME> -o jsonpath='{range .status.conditions[?
(@.type=="GracefulShutdownPostpone")]}{.type}{"\t"}{.status}{"\t"}{.reason}{"\t"}
{.message}{"\n"}{end}'
```

Possible values of the `reason` field:

- `WaitingForShutdownSignal`: The shutdown blocking mechanism is running on the node, but the blocking has not yet begun.
- `PodsWithLabelAreRunningOnNode`: The shutdown has been blocked because there are pods with the label `pod.deckhouse.io/inhibit-node-shutdown` still running on the node.
- `NoRunningPodsWithLabel`: The blocking has been released and the shutdown can continue.

How the fencing mechanism handles different node types?

The fencing mechanism is triggered when a node loses connectivity with the cluster and protects it from “zombie” nodes in an undefined state. To prevent short-time failures from causing cascading actions, the mechanism operates in two stages: first, it marks the node as suspicious (at the `memberlist/Lifeguard` gossip protocol level), and then isolates it — shuts the node down and, if necessary, deletes the associated Node object.

For more details on how the fencing-agent and fencing-controller work, refer to the description of the `spec.fencing.mode` parameter of the NodeGroup resource.

The behavior depends on the node type (the `node.deckhouse.io/type` label).

Cloud nodes (CloudEphemeral, CloudPermanent)

When fencing is triggered in `Watchdog` mode:

1. The fencing-agent stops resetting the watchdog timer (the `softdog` kernel module). When the timeout expires, a kernel panic occurs — this guarantees that no workloads remain running on the node.
2. The fencing-controller deletes the Node object from the cluster.
3. The cloud-provider-controller (MCM/CAPI) picks up the Node deletion, removes the corresponding virtual machine, and provisions a new one if needed. As a result, the failed node is recreated automatically.

Theoretically, a scenario is possible where kubelet on a “stuck” VM recovers before the cloud-provider deletes the machine and attempts to re-register the node in the cluster. The probability of this is minimal due to the following protecting mechanisms:

- A kernel panic leaves the node in a non-operational state. Manual VM restart is required for recovery.
- Automatic node reboot after a kernel panic is disabled at the OS level for all nodes with `fencing` enabled.
- There are safety delays between the kernel panic and the moment the cloud-provider processes the machine.

Static nodes (Static, CloudStatic)

For static nodes, the Node object is **not deleted**, since the underlying machine cannot be automatically recreated. In this case:

1. The fencing-agent shuts the node down via a kernel panic to prevent undefined behavior.
2. The fencing-controller only deletes pods from node.
3. The Node object remains in the cluster, and the node waits until it is manually restored by an operator.

Disabling automatic reboot after kernel panic

In all cases, when `fencing` is enabled, automatic node reboot after a kernel panic is disabled at the OS level (via kernel settings). This prevents a node from rejoining the cluster in an undefined state:

- **For cloud nodes**: Until the cloud-provider deletes or recreates the virtual machine.
- **For static nodes**: Until the operator restores the node manually.

If you observe a scenario where a node with `fencing` enabled returns to the cluster in a way that differs from the behavior described above, contact the support. This will help clarify the sequence of checks and protecting mechanisms.

How do I work with GPU nodes?

Step-by-step procedure for adding a GPU node to the cluster

Starting with Deckhouse 1.71, if a NodeGroup resource contains the `spec.gpu` section, the `node-manager` module **automatically**:

- Configures containerd with `default_runtime = "nvidia"`.
- Applies the required system settings (including fixes for the NVIDIA Container Toolkit).
- Deploys the following system components:
  - NFD
  - GFD
  - NVIDIA Device Plugin
  - DCGM Exporter
  - MIG Manager (if needed)

For the list of platforms supported by NVIDIA Container Toolkit, see the official documentation.

> **i** To ensure correct operation, clearly specify the GPU sharing strategy in the `spec.gpu.sharing` parameter (`Exclusive`, `TimeSlicing`, or `MIG`).
>
> Containerd on GPU nodes is configured automatically. Do not change its configuration manually (for example, via the NodeGroupConfiguration resource or a configuration on the node).

> **i** If a GPU is used in a single-node cluster and the `MIG` strategy is selected, an additional manual procedure is required after each node reboot to reapply the MIG configuration. For details, see MIG in a single-node cluster section.

To add a GPU node to the cluster, perform the following steps:

1. Create a NodeGroup resource for GPU nodes.

   An example with **TimeSlicing** enabled (`partitionCount: 4`) and typical `taints` and `labels`:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: gpu
spec:
  nodeType: CloudStatic  # Can also be Static or CloudEphemeral — depending on your
infrastructure.
  gpu:
    sharing: TimeSlicing
    timeSlicing:
      partitionCount: 4
  nodeTemplate:
    labels:
      node-role/gpu: ""
    taints:
    - key: node-role
      value: gpu
      effect: NoSchedule
```

> **i** If you use custom taint keys, ensure they are allowed in the ModuleConfig `global` in the array `.spec.settings.modules.placement.customTolerationKeys`, so that workloads can add the corresponding `tolerations`.

A full field schema is available in the NodeGroup custom resource reference.

2. Install the NVIDIA driver and NVIDIA Container Toolkit.

   Install the NVIDIA driver and NVIDIA Container Toolkit directly on the nodes, either manually or via the NodeGroupConfiguration resource. Current NodeGroupConfiguration examples for the `gpu` NodeGroup (Ubuntu, Debian, CentOS) are available in the `gpu` module FAQ.

3. After the configuration is applied, perform a bootstrap and reboot the nodes so that settings are applied and the drivers get installed.

4. Verify installation on the node using the command:

   ```
   nvidia-smi
   ```

   > **⚠** If the GPU node was prepared in advance, shipped with preinstalled NVIDIA drivers, or runs an OS with a graphical environment, make sure that the GPU is not being used by third-party processes.
   >
   > In the `nvidia-smi` output, the `Processes` section must not contain third-party processes using the GPU. On nodes with a graphical environment, these may include, for example, graphical session or display manager processes: `Xorg`, `gnome-shell`, `gdm`, `sddm`, `lightdm`, and others. Such processes can consume GPU memory and interfere with correct workload operation as well as MIG configuration.

   Expected proper output (example):

```
root@k8s-dvp-w1-gpu:~# nvidia-smi
Tue Aug  5 07:08:48 2025
+-----------------------------------------------------------------------------------------
----+
| NVIDIA-SMI 535.247.01             Driver Version: 535.247.01   CUDA Version: 12.2
|
|-----------------------------------------+----------------------+-----------------
----+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr.
ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute
M. |
|                                         |                      |               MIG
M. |
|=========================================+======================+=================
====|
|   0  Tesla V100-PCIE-32GB           Off | 00000000:65:00.0 Off |
0 |
| N/A   32C    P0              35W / 250W |      0MiB / 32768MiB |      0%
Default |
|                                         |                      |
N/A |
+-----------------------------------------+----------------------+-----------------
----+

+-----------------------------------------------------------------------------------------
----+
| Processes:
|
|  No running processes found
|
+-----------------------------------------------------------------------------------------
----+
```

5. Verify infrastructure components in the cluster.

   NVIDIA Pods in the `d8-nvidia-gpu` namespace:

   ```
   d8 k -n d8-nvidia-gpu get pod
   ```

   Expected proper output (example):

   ```
   NAME                                   READY   STATUS    RESTARTS   AGE
   gpu-feature-discovery-80ceb7d-r842q   2/2     Running   0          2m53s
   nvidia-dcgm-exporter-w9v9h            1/1     Running   0          2m53s
   nvidia-dcgm-njqqb                     1/1     Running   0          2m53s
   nvidia-device-plugin-80ceb7d-8xt8g    2/2     Running   0          2m53s
   ```

   NFD Pods in the `d8-cloud-instance-manager` namespace:

   ```
   d8 k -n d8-cloud-instance-manager get pods | egrep '^(NAME|node-feature-discovery)'
   ```

   Expected proper output (example):

   ```
   NAME                                          READY   STATUS      RESTARTS
   AGE
   node-feature-discovery-gc-6d845765df-45vpj       1/1     Running     0
   3m6s
   node-feature-discovery-master-74696fd9d5-wkjk4   1/1     Running     0
   3m6s
   node-feature-discovery-worker-5f4kv              1/1     Running     0
   3m8s
   ```

   Resource exposure on the node:

```
d8 k describe node <node-name>
```

Output snippet (example):

```
Capacity:
  cpu:                40
  memory:             263566308Ki
  nvidia.com/gpu:     4
Allocatable:
  cpu:                39930m
  memory:             262648294441
  nvidia.com/gpu:     4
```

6. Run functional tests.

   Option A. Invoke `nvidia-smi` from inside a container.

   1. Create a Job:

      ```
      apiVersion: batch/v1
      kind: Job
      metadata:
        name: nvidia-cuda-test
        namespace: default
      spec:
        completions: 1
        template:
          spec:
            restartPolicy: Never
            nodeSelector:
              node.deckhouse.io/group: gpu
              node-role/gpu: ""
            tolerations:
              - key: "node-role"
                operator: "Equal"
                value: "gpu"
                effect: "NoSchedule"
            containers:
              - name: nvidia-cuda-test
                image: nvidia/cuda:11.6.2-base-ubuntu20.04
                imagePullPolicy: "IfNotPresent"
                command:
                  - nvidia-smi
      ```

   2. Check the logs using the command:

      ```
      d8 k logs job/nvidia-cuda-test
      ```

      Output example:

```
Tue Aug  5 07:48:02 2025
+---------------------------------------------------------------------------
--------+
| NVIDIA-SMI 535.247.01             Driver Version: 535.247.01   CUDA Version:
12.2     |
|-----------------------------------------+----------------------+-----------
--------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util
Compute M. |
|                                         |                      |
MIG M. |
|=========================================+======================+==========
========|
|   0  Tesla V100-PCIE-32GB           Off | 00000000:65:00.0 Off |
0 |
| N/A   31C    P0              23W / 250W |      0MiB / 32768MiB |      0%
Default |
|                                         |                      |
N/A |
+-----------------------------------------+----------------------+-----------
--------+

+---------------------------------------------------------------------------
--------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                            GPU
Memory |
|        ID   ID
Usage      |
|=========================================================================
========|
|  No running processes found
|
+---------------------------------------------------------------------------
--------+
```

Option B. CUDA sample (vectoradd).

1. Create a Job:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: gpu-operator-test
  namespace: default
spec:
  completions: 1
  template:
    spec:
      restartPolicy: Never
      nodeSelector:
        node.deckhouse.io/group: gpu
      tolerations:
        - key: "node-role"
          operator: "Equal"
          value: "gpu"
          effect: "NoSchedule"
      containers:
        - name: gpu-operator-test
          image: nvidia/samples:vectoradd-cuda10.2
          imagePullPolicy: "IfNotPresent"
```

2. Check the logs using the command:

   ```
   d8 k logs job/gpu-operator-test
   ```

   Output example:

   ```
   [Vector addition of 50000 elements]
   Copy input data from the host memory to the CUDA device
   CUDA kernel launch with 196 blocks of 256 threads
   Copy output data from the CUDA device to the host memory
   Test PASSED
   Done
   ```

How to monitor GPUs?

Deckhouse Kubernetes Platform automatically deploys **DCGM Exporter**; GPU metrics are scraped by Prometheus and available in Grafana.

Which GPU modes are supported?

- **Exclusive** — the node exposes the `nvidia.com/gpu` resource; each Pod receives an entire GPU.
- **TimeSlicing** — time-sharing a single GPU among multiple Pods (default `partitionCount: 4`); Pods still request `nvidia.com/gpu`.
- **MIG (Multi-Instance GPU)** — hardware partitioning of supported GPUs into independent instances; with the `all-1g.5gb` profile the cluster exposes resources like `nvidia.com/mig-1g.5gb`.

See examples in Managing nodes: examples section.

How to view available MIG profiles in the cluster?

Pre-defined profiles are stored in the **`mig-parted-config`** ConfigMap inside the **`d8-nvidia-gpu`** namespace and can be viewed with the command:

```
d8 k -n d8-nvidia-gpu get cm mig-parted-config -o json | jq -r '.data["config.yaml"]'
```

The `mig-configs:` section lists the **GPU models (by PCI ID) and the MIG profiles each card supports**—for example `all-1g.5gb`, `all-2g.10gb`, `all-balanced`. Select the profile that matches your accelerator and set its name in `spec.gpu.mig.partedConfig` of the NodeGroup.

How to set a custom per-GPU MIG layout on a node?

Set the `partedConfig` parameter in NodeGroup to `custom` and define MIG partitioning per GPU index:

```
spec:
  gpu:
    sharing: MIG
    mig:
      partedConfig: custom
      customConfigs:
        - index: 0
          slices:
            - profile: 7g.80gb
              count: 1   # Can be in the range from 1 to 7.
        - index: 1
          slices:
            - profile: 3g.40gb
              count: 2
            - profile: 1g.10gb
              count: 1
        # Add more indexes as needed.
```

As a result, the `node-manager` module automatically:

- Adds a configuration named `custom-<node-group-name>-<hash>` into the `mig-parted-config` ConfigMap, where `<hash>` is calculated based on the partitioning scheme (long NodeGroup names are truncated with a hash suffix to fit into a label).
- Sets the label `nvidia.com/mig.config=custom-<node-group-name>-<hash>` to the nodes in the corresponding group.

A separate `custom-<ng>-<hash>` configuration is created for each group of nodes, and configuration names do not overlap.

MIG profile does not activate — what to check?

1. Check the GPU model. MIG is supported in the H100/A100/A30 models and **not** supported in V100/T4. To verify the support in a model, refer to profile tables in the NVIDIA MIG guide.
2. Ensure the GPU is not being used by OS processes or user applications. If a graphical environment, display manager, or other GPU-consuming processes are running on the node, applying the MIG configuration may fail or may not take effect until the GPU is released. Check this with the following command:

   ```
   nvidia-smi
   ```

3. Check the NodeGroup configuration:

   ```
   gpu:
     sharing: MIG
     mig:
       partedConfig: all-1g.5gb
   ```

4. Wait until `nvidia-mig-manager` completes the **drain** of the node and reconfigures the GPU.

   This process can take several minutes.

   While it is running, the node is tainted with `mig-reconfigure`. When the operation succeeds, that taint is removed.

5. Track the progress via the `nvidia.com/mig.config.state` label on the node:

   `pending`, `rebooting`, `success` (or `error` if something goes wrong).

6. If `nvidia.com/mig-*` resources are still missing, check:

   ```
   d8 k -n d8-nvidia-gpu logs daemonset/nvidia-mig-manager
   nvidia-smi -L
   ```

Are AMD or Intel GPUs supported?

At this time, Deckhouse Kubernetes Platform automatically configures **NVIDIA GPUs only**. Support for **AMD (ROCm)** and **Intel GPUs** is being worked on and is planned for future releases.

Examples

> **i** The module lifecycle stage: General Availability

Below are some examples of NodeGroup description, as well as installing the cert-manager plugin for `kubectl` and setting the `sysctl` parameter.

Examples of the `NodeGroup` configuration

Cloud nodes

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: test
spec:
  nodeType: CloudEphemeral
  cloudInstances:
    zones:
      - eu-west-1a
      - eu-west-1b
    minPerZone: 1
    maxPerZone: 2
    classReference:
      kind: AWSInstanceClass
      name: test
  nodeTemplate:
    labels:
      tier: test
```

Static nodes

Use `nodeType: Static` for physical servers and VMs on Hypervisors.

An example:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
```

Adding nodes to such a group is done manually using the pre-made scripts.

You can also use a method that adds static nodes using the Cluster API Provider Static.

System nodes

Below is an example of a system node group manifest.

When describing a NodeGroup with Static nodes, specify the value `Static` in the `nodeType` field and use the `staticInstances` field to describe the parameters for provisioning static machines to the cluster.

When describing a NodeGroup with CloudEphemeral type cloud nodes, specify the value `CloudEphemeral` in the `nodeType` field and use the `cloudInstances` field to describe the parameters for provisioning the cloud-based VMs.

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: system
spec:
  nodeTemplate:
    labels:
      node-role.deckhouse.io/system: ""
    taints:
      - effect: NoExecute
        key: dedicated.deckhouse.io
        value: system
  # Example for Static nodes.
  nodeType: Static
  staticInstances:
    count: 2
    labelSelector:
      matchLabels:
        role: system
  # Example for CloudEphemeral nodes.
  # nodeType: CloudEphemeral
  # cloudInstances:
  #   classReference:
  #     kind: YandexInstanceClass
  #     name: large
  #   maxPerZone: 2
  #   minPerZone: 1
  #   zones:
  #   - ru-central1-d
```

Nodes with GPU

GPU nodes require the **NVIDIA driver** and the **NVIDIA Container Toolkit**. There are two ways to install the driver:

1. **Manual installation** — the administrator installs the driver before the node joins the cluster.
2. **Automation via `NodeGroupConfiguration`** (more details in the Step-by-step procedure for adding a GPU node to the cluster section).

After the driver is detected and the NodeGroup includes the `spec.gpu` section, `node-manager` enables full GPU support by deploying **NFD**, **GFD**, **NVIDIA Device Plugin**, **DCGM Exporter**, and, if required, **MIG Manager**.

> **i** GPU nodes are usually tainted (e.g. `node-role=gpu:NoSchedule`) so that regular workloads don’t land there by accident. A workload that needs a GPU just adds the matching `tolerations` and `nodeSelector`.

See the full field reference in the NodeGroup custom resource documentation.

Below are examples of NodeGroup manifests for typical GPU operating modes (Exclusive, TimeSlicing, MIG).

Exclusive mode (one Pod — one GPU)

Each Pod gets an entire physical GPU; the cluster exposes the `nvidia.com/gpu` resource.

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: gpu-exclusive
spec:
  nodeType: Static
  gpu:
    sharing: Exclusive
  nodeTemplate:
    labels:
      node-role/gpu: ""
    taints:
    - key: node-role
      value: gpu
      effect: NoSchedule
```

Time-slicing (4 partitions)

The GPU is time-sliced: up to four Pods can share one card sequentially. Suitable for experiments, CI, and light inference workloads.

Pods still request the `nvidia.com/gpu` resource.

```
spec:
  gpu:
    sharing: TimeSlicing
    timeSlicing:
      partitionCount: 4
```

MIG (`all-1g.5gb` profile)

A hardware-partitioned GPU (A100, A30, etc.) is split into independent instances. The scheduler exposes resources like `nvidia.com/mig-1g.5gb`.

For a complete list of supported GPUs and their profiles you can see it by using instruction.

```
spec:
  gpu:
    sharing: MIG
    mig:
      partedConfig: all-1g.5gb
```

Smoke-test Job (CUDA vectoradd)

```
apiVersion: batch/v1
kind: Job
metadata:
  name: cuda-vectoradd
spec:
  template:
    spec:
      restartPolicy: OnFailure
      nodeSelector:
        node-role/gpu: ""
      tolerations:
      - key: node-role
        value: gpu
        effect: NoSchedule
      containers:
      - name: cuda-vectoradd
        image: nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubuntu20.04
        resources:
          limits:
            nvidia.com/gpu: 1
```

This Job runs NVIDIA's **vectoradd** CUDA sample. If the Pod finishes with `Succeeded`, the GPU is present and configured correctly.

Adding a static node to a cluster

Adding a static node can be done manually or using the Cluster API Provider Static.

Manually

Follow the steps below to add a new static node (e.g., VM or bare metal server) to the cluster:

1. For CloudStatic nodes in the following cloud providers, refer to the steps outlined in the documentation:
   - For AWS
   - For GCP
   - For YC
2. Use the existing one or create a new NodeGroup custom resource (see the example for the `NodeGroup` called `worker`). The nodeType parameter for static nodes in the NodeGroup must be `Static` or `CloudStatic`.
3. Get the Base64-encoded script code to add and configure the node.

   Here is how you can get Base64-encoded script code to add a node to the `worker` NodeGroup:

   ```
   NODE_GROUP=worker
   d8 k -n d8-cloud-instance-manager get secret manual-bootstrap-for-${NODE_GROUP} -o json | jq
   '.data."bootstrap.sh"' -r
   ```
4. Pre-configure the new node according to the specifics of your environment. For example:
   - Add all the necessary mount points to the `/etc/fstab` file (NFS, Ceph, etc.).
   - Install the necessary packages.
   - Configure network connectivity between the new node and the other nodes of the cluster.
5. Connect to the new node over SSH and run the following command, inserting the Base64 string you got in step 3:

   ```
   echo <Base64-CODE> | base64 -d | bash
   ```

Using the Cluster API Provider Static

> ⚠ If you have previously increased the number of master nodes in the cluster in the NodeGroup `master` (parameter `spec.staticInstances.count`), before adding regular nodes using CAPS, make sure that they will not be "captured".

A brief example of adding a static node to a cluster using Cluster API Provider Static (CAPS):

1. Prepare the necessary resources.

- Allocate a server (or a virtual machine), configure networking, etc. If required, install specific OS packages and add the mount points on the node.
- Create a user (`caps` in the example below) and add it to sudoers by running the following command **on the server**:

  ```
  useradd -m -s /bin/bash caps
  usermod -aG sudo caps
  ```

- Allow the user to run sudo commands without having to enter a password. For this, add the following line to the sudo configuration **on the server** (you can either edit the `/etc/sudoers` file, or run the `sudo visudo` command, or use some other method):

  ```
  caps ALL=(ALL) NOPASSWD: ALL
  ```

- Generate a pair of SSH keys with an empty passphrase **on the server**:

  ```
  ssh-keygen -t  -f caps-id -C "" -N ""
  ```

  The public and private keys of the `caps` user will be stored in the `caps-id.pub` and `caps-id` files in the current directory on the server.

- Add the generated public key to the `/home/caps/.ssh/authorized_keys` file of the `caps` user by executing the following commands in the keys directory **on the server**:

  ```
  mkdir -p /home/caps/.ssh
  cat caps-id.pub >> /home/caps/.ssh/authorized_keys
  chmod 700 /home/caps/.ssh
  chmod 600 /home/caps/.ssh/authorized_keys
  chown -R caps:caps /home/caps/
  ```

2. Create a SSHCredentials resource in the cluster:

   Run the following command in the user key directory **on the server** to encode the private key to Base64:

   ```
   base64 -w0 caps-id
   ```

   On any computer with `kubectl` configured to manage the cluster, create an environment variable with the value of the Base64-encoded private key you generated in the previous step:

   ```
   CAPS_PRIVATE_KEY_BASE64=<BASE64-ENCODED PRIVATE KEY>
   ```

   Create a `SSHCredentials` resource in the cluster (note that from this point on, you have to use `kubectl` configured to manage the cluster):

   ```
   d8 k create -f - <<EOF
   apiVersion: deckhouse.io/v1alpha1
   kind: SSHCredentials
   metadata:
     name: credentials
   spec:
     user: caps
     privateSSHKey: "${CAPS_PRIVATE_KEY_BASE64}"
   EOF
   ```

3. Create a StaticInstance resource in the cluster; specify the IP address of the static node server:

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  # Specify the IP address of the static node server.
  address: "<SERVER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
EOF
```

4. Create a NodeGroup resource in the cluster. Value of `count` defines number of `staticInstances` which fall under the `labelSelector` that will be bootstrapped and joined into the `nodeGroup`, in this example this is `1`:

> **i** The `labelSelector` field in the `NodeGroup` resource is immutable. To update the `labelSelector`, you need to create a new `NodeGroup` and move the static nodes into it by changing their labels.

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: worker
EOF
```

> **i** If it is necessary to add nodes to an existing node group, specify the desired number in the `.spec.count` field of the NodeGroup.

Using Cluster API Provider Static for multiple node groups

This example shows how you can use filters in the StaticInstance label selector to group static nodes and use them in different NodeGroups. Here, two node groups (`front` and `worker`) are used for different tasks. Each group includes nodes with different characteristics — the `front` group has two servers and the `worker` group has one.

1. Prepare the required resources (3 servers or virtual machines) and create the `SSHCredentials` resource in the same way as step 1 and step 2 of the example.
2. Create two NodeGroup in the cluster (from this point on, use `kubectl` configured to manage the cluster):

> **i** The `labelSelector` field in the `NodeGroup` resource is immutable. To update the `labelSelector`, you need to create a new `NodeGroup` and move the static nodes into it by changing their labels.

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: front
spec:
  nodeType: Static
  staticInstances:
    count: 2
    labelSelector:
      matchLabels:
        role: front
---
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: worker
EOF
```

3. Create StaticInstance resources in the cluster and specify the valid IP addresses of the servers:

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-front-1
  labels:
    role: front
spec:
  address: "<SERVER-FRONT-IP1>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
---
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-front-2
  labels:
    role: front
spec:
  address: "<SERVER-FRONT-IP2>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
---
apiVersion: deckhouse.io/v1alpha1
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  address: "<SERVER-WORKER-IP>"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
EOF
```

Cluster API Provider Static: Moving Instances Between Node Groups

⚠ During the process of transferring instances between node groups, the instance will be cleaned and re-bootstrapped, and the `Node` object will be recreated.

This section describes the process of moving static instances between different node groups (NodeGroup) using the Cluster API Provider Static (CAPS). The process involves modifying the NodeGroup configuration and updating the labels of the corresponding StaticInstance.

Initial Configuration

Assume that there is already a NodeGroup named `worker` in the cluster, configured to manage one static instance with the label `role: worker`.

`NodeGroup` worker:

```
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: worker
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: worker
```

`StaticInstance` static-worker-1:

```
apiVersion: deckhouse.io/v1alpha2
kind: StaticInstance
metadata:
  name: static-worker-1
  labels:
    role: worker
spec:
  address: "192.168.1.100"
  credentialsRef:
    kind: SSHCredentials
    name: credentials
```

Steps to Move an Instance Between Node Groups

1. Create a New `NodeGroup` for the Target Node Group

Create a new NodeGroup resource, for example, named `front`, which will manage a static instance with the label `role: front`.

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: NodeGroup
metadata:
  name: front
spec:
  nodeType: Static
  staticInstances:
    count: 1
    labelSelector:
      matchLabels:
        role: front
EOF
```

2. Update the Label on the `StaticInstance`

Change the `role` label of the existing StaticInstance from `worker` to `front`. This will allow the new NodeGroup `front` to manage this instance.

```
d8 k label staticinstance static-worker-1 role=front --overwrite
```

3. Decrease the Number of Static Instances in the Original `NodeGroup`

Update the NodeGroup resource `worker` by reducing the `count` parameter from `1` to `0`.

```
d8 k patch nodegroup worker -p '{"spec": {"staticInstances": {"count": 0}}}' --type=merge
```

An example of the `NodeUser` configuration

```
apiVersion: deckhouse.io/v1
kind: NodeUser
metadata:
  name: testuser
spec:
  uid: 1100
  sshPublicKeys:
    - "<SSH_PUBLIC_KEY>"
  passwordHash: <PASSWORD_HASH>
  isSudoer: true
```

An example of the `NodeGroupConfiguration` configuration

Installing the cert-manager plugin for kubectl on master nodes

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-cert-manager-plugin.sh
spec:
  weight: 100
  bundles:
  - "*"
  nodeGroups:
  - "master"
  content: |
    if [ -x /usr/local/bin/kubectl-cert_manager ]; then
      exit 0
    fi
    curl -L https://github.com/cert-manager/cert-
manager/releases/download/v1.7.1/kubectl-cert_manager-linux-amd64.tar.gz -o - | tar -zxvf
- kubectl-cert_manager
    mv kubectl-cert_manager /usr/local/bin
```

Tuning sysctl parameters

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: sysctl-tune.sh
spec:
  weight: 100
  bundles:
  - "*"
  nodeGroups:
  - "*"
  content: |
    sysctl -w vm.max_map_count=262144
```

Adding a root certificate to the host

⚠ Example is given for Ubuntu OS.
The method of adding certificates to the store may differ depending on the OS.

Change the bundles and content parameters to adapt the script to a different OS.

⚠ To use the certificate in `containerd` (including pulling containers from a private repository), a restart of the service is required after adding the certificate.

```
apiVersion: deckhouse.io/v1alpha1
kind: NodeGroupConfiguration
metadata:
  name: add-custom-ca.sh
spec:
  weight: 31
  nodeGroups:
  - '*'
  bundles:
  - 'ubuntu-lts'
  content: |-
    CERT_FILE_NAME=example_ca
    CERTS_FOLDER="/usr/local/share/ca-certificates"
    CERT_CONTENT=$(cat <<EOF
    -----BEGIN CERTIFICATE-----
    MIIDSjCCAjKgAwIBAgIRAJ4RR/WDuAym7M11JA8W7D0wDQYJKoZIhvcNAQELBQAw
    JTEjMCEGA1UEAxMabmV4dXMuNTEuMjUwLjQxLjIuc3NsaXAuaW8wHhcNMjQwODAx
    MTAzMjA4WhcNMjQxMDMwMTAzMjA4WjAlMSMwIQYDVQQDExpuZXh1cy41MS4yNTAu
    NDEuMi5zc2xpcC5pbzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAL1p
    WLPr2c4SZX/i4IS59Ly1USPjRE21G4pMYewUjkSXnYv7hUkHvbNL/P9dmGBm2Jsl
    WFlRZbzCv7+5/J+9mPVL2TdTbWuAcTUyaG5GZ/1w64AmAWxqGMFx4eyD1zo9eSmN
    G2jis8VofL9dWDfUYhRzJ90qKxgK6k7tfhL0pv7IHDbqf28fCEnkvxsA98lGkq3H
    fUfvHV6Oi8pcyPZ/c8ayIf4+JOnf7oW/TgWqI7x6R1CkdzwepJ8oU7PGc0ySUWaP
    G5bH3ofBavL0bNEsyScz4TFCJ9b4aO5GFAOmgjFMMUi9qXDH72sBSrgi08Dxmimg
    Hfs198SZr3br5GTJoAkCAwEAAaN1MHMwDgYDVR0PAQH/BAQDAgWgMAwGA1UdEwEB
    /wQCMAAwUwYDVR0RBEwwSoIPbmV4dXMuc3ZjLmxvY2FsghpuZXh1cy41MS4yNTAu
    NDEuMi5zc2xpcC5pb4IbZG9ja2VyLjUxLjI1MC40MS4yLnNzbGlwLmlvMA0GCSqG
    SIb3DQEBCwUAA4IBAQBvTjTTXWeWtfaUDrcp1YW1pKgZ7lTb27f3QCxukXpbC+wL
    dcb4EP/vDf+UqCogKl6rCEA0i23Dtn85KAE9PQZFfI5hLulptdOgUhO3Udluoy36
    D4WvUoCfgPgx12FrdanQBBja+oDsT1QeOpKwQJuwjpZcGfB2YZqhO0UcJpC8kxtU
    by3uoxJoveHPRlbM2+ACPBPlHu/yH7st24sr1CodJHNt6P8ugIBAZxi3/Hq0wj4K
    aaQzdGXeFckWaxIny7F1M3cIWEXWzhAFnoTgrwlklf7N7VWHPIvlIh1EYASsVYKn
    iATq8C7qhUOGsknDh3QSpOJeJmpcBwln11/9BGRP
    -----END CERTIFICATE-----
    EOF
    )

    # bb-event           - Creating subscription for event function. More information:
http://www.bashbooster.net/#event
    ## ca-file-updated   - Event name
    ## update-certs      - The function name that the event will call

    bb-event-on "ca-file-updated" "update-certs"

    update-certs() {          # Function with commands for adding a certificate to the
store
      update-ca-certificates
    }

    # bb-tmp-file - Creating temp file function. More information:
http://www.bashbooster.net/#tmp
    CERT_TMP_FILE="$( bb-tmp-file )"
    echo -e "${CERT_CONTENT}" > "${CERT_TMP_FILE}"

    # bb-sync-file                                - File synchronization function. More
information: http://www.bashbooster.net/#sync
    ## "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt"    - Destination file
    ##  ${CERT_TMP_FILE}                          - Source file
    ##  ca-file-updated                           - Name of event that will be called if
the file changes.

    bb-sync-file \
      "${CERTS_FOLDER}/${CERT_FILE_NAME}.crt" \
      ${CERT_TMP_FILE} \
      ca-file-updated
```

Adding the ability to download images from insecure container registry to containerd

The ability to download images from an insecure container registry is enabled using the `insecure_skip_verify` parameter in the containerd configuration file. For more information, see the How to add configuration for an additional registry.

The okmeter module

> **i** The module lifecycle stage: General Availability

The `okmeter` module installs the Okmeter agent as a `daemonset` in the `d8-okmeter` namespace, and deletes `okmeter` installed manually.

Configuration

> **i** The module lifecycle stage: General Availability

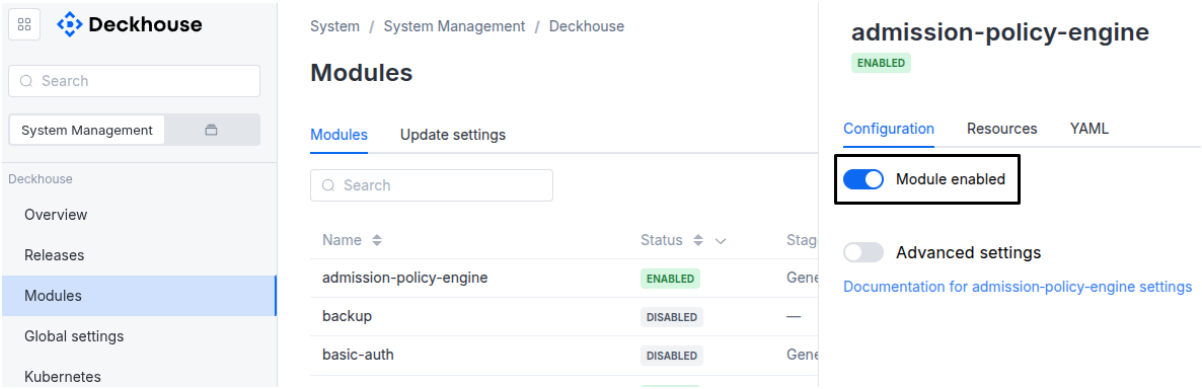
The module has 1 alert.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `okmeter` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable okmeter
```

- **Using ModuleConfig `okmeter`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `okmeter` (create it if necessary);

  Example of a manifest to enable module `okmeter`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: okmeter
spec:
  enabled: true
```

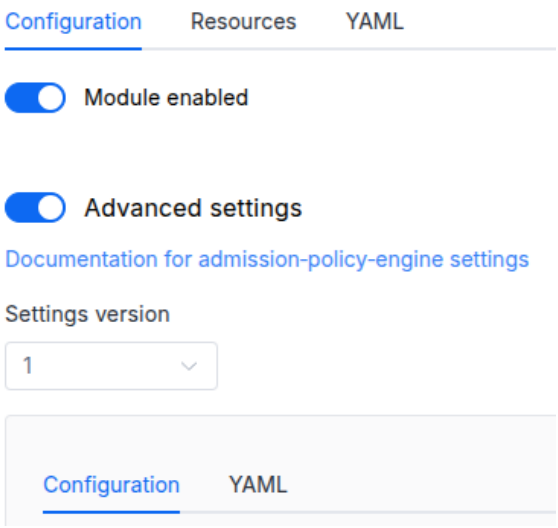
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `okmeter` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

Example:



You can also edit the ModuleConfig object `okmeter` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `okmeter`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `okmeter` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc okmeter
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `okmeter` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `okmeter`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: okmeter
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.apiKey**
    string

    Required value

    Key to access Okmeter API.

You can get the key from your personal Okmeter account (`OKMETER_API_TOKEN`).

Example:

```
apiKey: 5ff9z2a3-9127-1sh4-2192-06a3fc6e13e3
```

- **settings.disableAutoDiscovery**

  Disables okagent autodiscovery. Setting this option to `true` will leave only basic host monitoring, all other plugins (postgresql/mysql/redis/etc.) will be disabled. Default value is `false`

- **settings.image**
  object
  - **settings.image.repository**
    string

    Required value

    Docker repository for okmeter agent image.

    Default: `registry.okmeter.io/agent/okagent`

  - **settings.image.tag**
    string

    Required value

    Okmeter agent image tag.

    Default: `latest`

- **settings.nodeSelector**
  object

  The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  nodeSelector:
    test: test
  ```

- **settings.tolerations**
  array of objects

  The same as in the pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

Examples

> **i** The module lifecycle stage: General Availability

Example

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: okmeter
spec:
  version: 1
  enabled: true
  settings:
    apiKey: 5ff9z2a3-9127-1sh4-2192-06a3fc6e13e3
```

The openvpn module

> **i** The module lifecycle stage: General Availability

> **i** Web interfaces associated with the module: openvpn-admin

The openvpn module allows peers to authenticate each other using certificates and provides a simple web interface.

Using the web interface, you can:

- Issue certificates.
- Revoke certificates.
- Cancel certificate revocation.
- Get a ready-to-use custom configuration file.

Integrating with the user-authn module lets you manage user access to the web interface.

Exposing the VPN service

1. Select one or more external IP addresses for connection.
2. Use one of the connection methods:
   - By external IP address (`ExternalIP`) - if there are nodes with public IP addresses.
   - Using `LoadBalancer` - for all cloud providers and their placement schemes that support LoadBalancer ordering.
   - `Direct` - configure the traffic path manually: from the entry point to the cluster to the pod with OpenVPN.

Available cluster resources after connecting to the VPN

The following parameters are pushed to the user's computer after connecting to the VPN:

- The `kube-dns` address is added to the client's DNS servers to allow direct access to Kubernetes services via FQDN.
- A route to the local network.
- A route to the cluster service network.
- A route to the Pod network.

User's traffic audit

The module can log user's activity via VPN in JSON format. Traffic is grouped by `src_ip`, `dst_ip`, `src_port`, `dst_port`, `ip_proto` fields. Using the log-shipper module, container logs can be collected and stored for later auditing.

Third-party components

List of third-party software used in the `openvpn` module:

- OpenVPN 2.6.12

  License: Mixed

  A secure tunneling daemon.

- OpenSSL 3.3.2

  License: Apache License 2.0

  TLS/SSL and crypto library.

- Libcap-ng 0.8.5

  License: GNU General Public License v2.0

  Libcap-ng is a library for Linux that makes using posix capabilities easy..

Configuration

> **i** The module lifecycle stage: General Availability

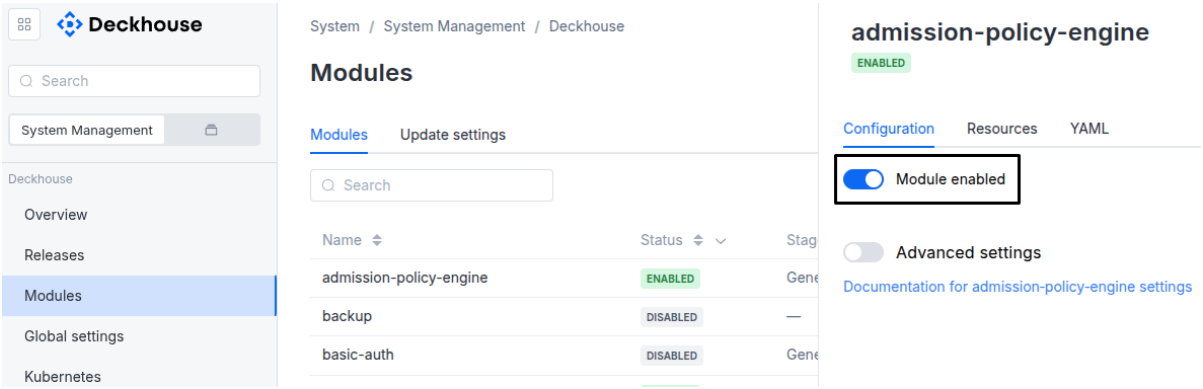
The module has 9 alerts.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `openvpn` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable openvpn
```

- **Using ModuleConfig `openvpn`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `openvpn` (create it if necessary);

  Example of a manifest to enable module `openvpn`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  enabled: true
```

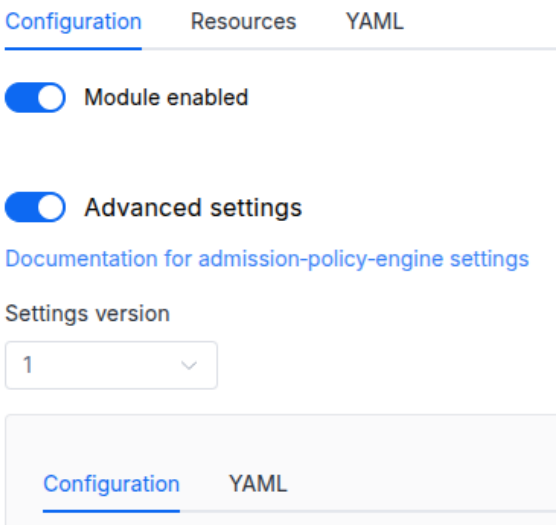
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `openvpn` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `openvpn` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `openvpn`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `openvpn` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc openvpn
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `openvpn` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `openvpn`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: openvpn
  spec:
    version: 2
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Conversion description is missing.

  Actions performed by the conversion (jq syntax)...

  - `del(.auth.password) | if .auth == {} then del(.auth) end`
  - `del(.storageClass)`

Parameters

Schema version: 2

- **settings**
  object
  - **settings.auth**
    object

    Options related to authentication or authorization in the application.

    - **settings.auth.allowedUserEmails**
      array of strings

      An array of emails of users that can access module's public web interfaces.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

    - **settings.auth.allowedUserGroups**
      array of strings

      An array of user groups that can access the OpenVPN admin panel.

      This parameter is used if the user-authn module is enabled or the externalAuthentication parameter is set.

      > **Caution!** Note that you must add those groups to the appropriate field in the DexProvider config if this module is used together with the user-authn one.

    - **settings.auth.externalAuthentication**
      object

      Parameters to enable external authentication based on the Ingress NGINX external-auth mechanism that uses the Nginx auth_request module.

      > **Note!** External authentication is enabled automatically if the user-authn module is enabled.

      - **settings.auth.externalAuthentication.authSignInURL**
        string

        The URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response code).

      - **settings.auth.externalAuthentication.authURL**
        string

        The URL of the authentication service. If the user is authenticated, the service should return an HTTP 200 response code.

    - **settings.auth.whitelistSourceRanges**
      array of strings

      The CIDR range for which authentication to access the OpenVPN is allowed.

  - **settings.clientCertExpirationDays**
    integer

    Certificate client expired period for OpenVPN in days. Specifies the number of days for which a client certificate generated using EasyRSA will be issued.

    Client certificate lifetime must be at least 1 day and no more than 10 years.

    Default: `3650`

    Allowed values: `1 <= X <= 3650`

  - **settings.externalHost**
    string

    An IP address or a domain clients use to connect to the OpenVPN server.

    By default, data from an `openvpn-external` service are used.

  - **settings.externalIP**
    string

    The IP address of a cluster node to connect OpenVPN clients.

It is only required if the `ExternalIP` inlet is used.

- **settings.externalPort**
  integer

  The port to expose on the `externalIP` or load balancer.

  Default: `5416`

- **settings.highAvailability**
  boolean

  Manually enable the high availability mode.

  By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

- **settings.hostPort**
  integer

  Port to connect to the OpenVPN server, which will be available on the node where it is scheduled.

  The parameter is available when selecting inlet `HostPort`.

  Default: `5416`

- **settings.https**
  object

  What certificate type to use with the OpenVPN admin panel.

  This parameter completely overrides the `global.modules.https` settings.

  - **settings.https.certManager**
    object
    - **settings.https.certManager.clusterIssuerName**
      string

      What ClusterIssuer to use for the OpenVPN admin panel (currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available; also, you can define your own).

      Currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available. Also, you can define your own.

      Default: `letsencrypt`

  - **settings.https.customCertificate**
    object
    - **settings.https.customCertificate.secretName**
      string

      The name of the Secret in the `d8-system` namespace to use with the OpenVPN admin panel (this Secret must have the kubernetes.io/tls format).

      Default: `false`

  - **settings.https.mode**
    string

    The HTTPS usage mode:

    - `CertManager` — the OpenVPN admin panel will use HTTPS and get a certificate from the ClusterIssuer defined in the `certManager.clusterIssuerName` parameter.
    - `CustomCertificate` — the OpenVPN admin panel will use the certificate from the `d8-system` namespace for HTTPS.
    - `Disabled` — in this mode, the OpenVPN admin panel works over HTTP only.
    - `OnlyInURI` — the OpenVPN admin panel will work over HTTP (thinking that there is an external HTTPS load balancer in front of it that terminates HTTPS traffic). All the links in the user-authn will be generated using the HTTPS scheme. Load balancer should provide a redirect from HTTP to HTTPS.

    Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller used for the OpenVPN admin panel.

  By default, the `modules.ingressClass` global value is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.inlet**
  string

  The way the connection is implemented.

  The following inlet types are supported:

  - `ExternalIP` — when there are nodes with public IPs. It is used together with the `externalIP` parameter.
  - `LoadBalancer` — for all cloud providers and cloud-based placement strategies that support the provision of LoadBalancers.
  - `HostPort` — the port of the OpenVPN server will be available on the node where it is scheduled. The port can be configured in the `hostPort` parameter.
  - `Direct` — for non-standard cases. You need to create a service called `openvpn-external` in the `d8-openvpn` namespace. It will route traffic to the Pod with the `app: openvpn` label to the port called `ovpn-tcp` (or just 1194). This service provides the externalIP, the IP address of the balancer or its host. If none of these are present, you need to specify the `externalHost` parameter.

  Allowed values: `ExternalIP`, `LoadBalancer`, `HostPort`, `Direct`

- **settings.loadBalancer**
  object

  A section of optional parameters of the `LoadBalancer` inlet.

  - **settings.loadBalancer.annotations**
    object

    Annotations to assign to the service for flexible configuration of the load balancer.

    > **i** **Note** that module does not take into account the specifics of setting annotations in different clouds. If annotations for the provision of the load balancer are only used when the service is being created, then you need to restart the module (disable/enable it) to update them.

  - **settings.loadBalancer.sourceRanges**
    array of strings

    A list of CIDRs that are allowed to connect to the Load Balancer.

    The cloud provider may not support this option or ignore it.

- **settings.nodeSelector**
  object

  The same as in the Pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

- **settings.pmacctEnabled**
  boolean

  Enable logging of user's activity via VPN in JSON format.

  All connections within `tun` interfaces will be collected via `libpcap` and logged as `{"event type": "purge", "ip src": "172.25.175.10", "ip dst": "10.222.0.10", "port_src": 32172, "port_dst": 53, "ip_proto": "udp", "packets": 1, "bytes": 53}`

  Default: `false`

- **settings.pushDefaultGatewayToClient**
  boolean

  Routing all client traffic through the VPN. A default route will be added to clients.

  Default: `false`

- **settings.pushToClientDNS**
  string

  The IP address of the DNS server to send to clients upon connection.

  By default, the IP address of the `kube-system/kube-dns` service is used.

- **settings.pushToClientRoutes**
  array of strings

  A list of routes to send to clients upon their connection.

By default, this list is generated automatically using the local cluster network, service subnet, and Pod subnet.

Example:

```
pushToClientRoutes:
- 172.25.176.0/24
- 172.25.178.0/255.255.255.0
```

- **settings.pushToClientSearchDomains**
  array of strings

  A list of search domains to send to clients upon connection.

  The default value is a value from the `global.discovery.clusterDomain` variable.

- **settings.tcpEnabled**
  boolean

  Default: `true`

- **settings.tolerations**
  array of objects

  The same as in the Pods' `spec.tolerations` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

- **settings.tunnelNetwork**
  string

  Tunnel network for OpenVPN server.

  Default: `172.25.175.0/24`

  Examples:

  ```
  tunnelNetwork: 172.25.175.0/24
  ```

  ```
  tunnelNetwork: 172.25.178.0/255.255.255.0
  ```

- **settings.udpEnabled**
  boolean

  Default: `false`

Authentication

user-authn module provides authentication by default. You can also configure authentication using the externalAuthentication parameter. If these options are disabled, the module will use basic auth with the auto-generated password.

To view the generated password, run the command:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values openvpn -o json | jq '.openvpn.internal.auth.password'
```

To generate a new password, delete the Secret:

```
d8 k -n d8-openvpn delete secret/basic-auth
```

> **i** The `auth.password` parameter is deprecated.

FAQ

> The module lifecycle stage: General Availability

Why doesn’t automatic DNS server configuration work when connecting to macOS and Linux using the OpenVPN client

Due to the architectural features of the Linux and macOS operating systems, automatic DNS server configuration when using the official OpenVPN client is not possible.

To configure the DNS server on such operating systems, the service provides the ability to use third-party scripts that run when the client connects or disconnects.

In the client configurations generated by the module, the blocks responsible for these settings are predefined and commented out:

```
# Uncomment the lines below for use with Linux
#script-security 2
# If you use resolved
#up /etc/openvpn/update-resolv-conf
#down /etc/openvpn/update-resolv-conf
# If you use systemd-resolved, first install the openvpn-systemd-resolved package
#up /etc/openvpn/update-systemd-resolved
#down /etc/openvpn/update-systemd-resolved
```

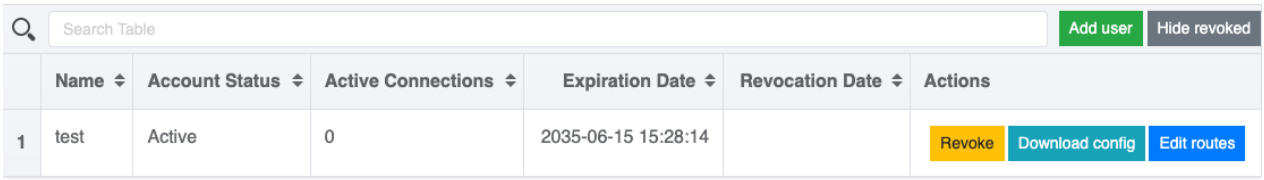
To use these code blocks, uncomment them (remove the initial `#` symbol), and specify the correct paths to the scripts.

You can either prepare the scripts yourself or use ready-made scripts from the official OpenVPN Community (for Linux). For macOS, you can use a third-party script.

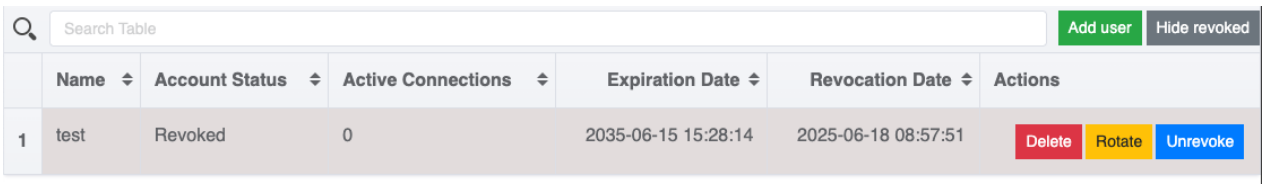
> Scripts must have execution permissions.

How to revoke, rotate or delete a user certificate

All operations with client certificates are performed via the openvpn-admin web interface. Buttons for managing certificates are available to the right of each user’s name:



To rotate (issue a new certificate) or delete a client, you must first revoke their current certificate:



Once revoked, the Renew (rotation) and Delete (removal) actions become available.

How to rotate a server certificate

The server certificate is rotated automatically several days before it expires.

If you need to rotate the certificate manually (e.g., due to certificate corruption or an unscheduled replacement), follow these steps:

1. Delete the secret `openvpn-pki-server` in the namespace `d8-openvpn`:

   ```
   d8 k -n d8-openvpn delete secrets openvpn-pki-server
   ```

2. Restart the OpenVPN pods to trigger the generation of a new certificate:

   ```
   d8 k -n d8-openvpn rollout restart sts openvpn
   ```

A new certificate will be generated automatically when the pods start.

How to rotate a root certificate (CA)

The root certificate (CA) and server certificate are rotated automatically 1 day before expiration. Automatic rotation of clients certificates is not provided. The root certificate (CA) is used to sign all certificates in OpenVPN — both server and client. Therefore, when replacing the CA, you must reissue all dependent certificates.

Steps to rotate the root certificate:

1. Revoke or delete all active client certificates using the openvpn-admin web interface. If you choose to revoke certificates, you can rotate them later (Renew) after the CA is replaced, without recreating the clients.
2. Delete secrets `openvpn-pki-ca` and `openvpn-pki-server` in the namespace `d8-openvpn`:

   ```
   d8 k -n d8-openvpn delete secrets openvpn-pki-ca openvpn-pki-server
   ```

3. Restart OpenVPN pods:

   ```
   d8 k -n d8-openvpn rollout restart sts openvpn
   ```

4. Rotate certificates of revoked users, or create new ones.
5. Delete all revoked certificate secrets:

   ```
   d8 k -n d8-openvpn delete secrets -l revokedForever=true
   ```

Examples

> **i** The module lifecycle stage: General Availability

An example for bare metal clusters

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
   inlet: ExternalIP
   externalIP: 5.4.54.4
```

An example for AWS & Google Cloud

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    inlet: LoadBalancer
```

An example for an external load balancer with a public IP address

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: openvpn
spec:
  version: 2
  enabled: true
  settings:
    externalHost: 5.4.54.4
    externalIP: 192.168.0.30 # The internal IP address to forward the external LB's
traffic to.
    inlet: ExternalIP
    nodeSelector:
      kubernetes.io/hostname: node
```

The operator-prometheus module

> **i** The module lifecycle stage: General Availability

The module installs the Prometheus Operator, which automates the deployment and management of Prometheus installations.

Operator:

- Creates the following custom resources:
  - Prometheus — describes a Prometheus installation (cluster);
  - ServiceMonitor — defines how to collect metrics from services;
  - Alertmanager — describes an Alertmanager cluster;
  - PrometheusRule — contains a set of Prometheus rules.
- Watches these resources and:
  - Generates a StatefulSet with Prometheus and the required configuration files, storing them in a Secret.
  - When ServiceMonitor or PrometheusRule change, updates the Prometheus configuration by modifying the corresponding Secret.

Prometheus

Prometheus functions

The Prometheus server performs two primary tasks — it collects metrics and processes rules.

- For each monitoring target, at every `scrape_interval` an HTTP request is sent to that target. The metrics in its own text format are stored in the Prometheus database.
- At every `evaluation_interval`, rules are processed, resulting in:
  - alerts being sent;
  - new metrics (rule evaluation results) being written to the database.

Prometheus configuration

Prometheus configuration consists of the main `config` file and rule files (`rule files`).

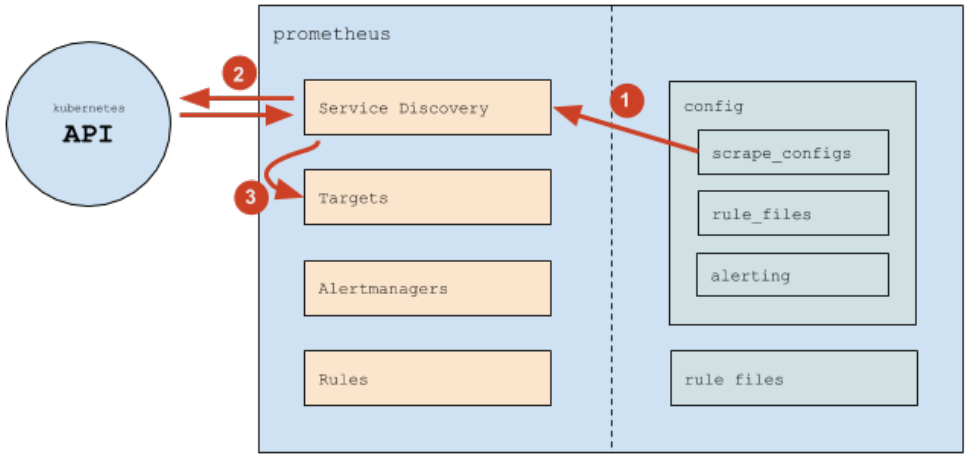
The key sections used in `config` are:

- `scrape_configs`: Defines where and how to discover monitoring targets.
- `rule_files`: Lists the paths to rule files that Prometheus should load. Example:

  ```yaml rule_files:
  - /etc/prometheus/rules/rules-0/*
  - /etc/prometheus/rules/rules-1/*
  ```
- `alerting`: Defines how to discover and where to send alerts to Alertmanager clusters. Structurally, this section is similar to `scrape_configs`, but its result is a list of endpoints to which Prometheus sends notifications.

Sources of monitoring targets

General Prometheus workflow:



- Prometheus reads the `scrape_configs` section and configures its internal Service Discovery.
- Service Discovery interacts with the Kubernetes API (retrieving endpoints and related metadata).
- Based on cluster events, Service Discovery updates the list of monitoring targets.

In `scrape_configs`, a set of `scrape job`s is defined — logical descriptions of where and how to collect metrics. Example:

```
scrape_configs:
  # General settings.
  - job_name: d8-monitoring/custom/0   # Name of the scrape job, shown in the Service
Discovery section.
    scrape_interval: 30s              # How often to scrape data.
    scrape_timeout: 10s               # Request timeout.
    metrics_path: /metrics             # HTTP path.
    scheme: http                      # HTTP or HTTPS.
    # Service Discovery settings.
    kubernetes_sd_configs:             # Get targets from Kubernetes.
    - api_server: null                # API server address is taken from environment
variables (present in every Pod).
      role: endpoints                 # Use endpoints as targets.
      namespaces:
        names:                        # Restrict the list of namespaces.
        - foo
        - baz
    # Filtering settings (which endpoints to include/exclude) and relabeling (which labels
to add or remove; applies to all scraped metrics).
    relabel_configs:
    # Filter by the value of the prometheus_custom_target label (taken from the Service
associated with the endpoint).
    - source_labels: [__meta_kubernetes_service_label_prometheus_custom_target]
      regex: .+                       # Any non-empty label.
      action: keep
    # Filter by port name.
    - source_labels: [__meta_kubernetes_endpointslice_port_name]
      regex: http-metrics              # Only the port named http-metrics.
      action: keep
    # Add the job label. Use the value of the prometheus_custom_target label on the
Service, prefixed with "custom-".
    #
    # The job label:
    #    * defines the group name under which the target will appear;
    #    * is added to metrics for convenient filtering in rules and dashboards.
    - source_labels: [__meta_kubernetes_service_label_prometheus_custom_target]
      regex: (.*)
      target_label: job
      replacement: custom-$1
      action: replace
    # Add the namespace label.
    - source_labels: [__meta_kubernetes_namespace]
      regex: (.*)
      target_label: namespace
      replacement: $1
      action: replace
    # Add the service label.
    - source_labels: [__meta_kubernetes_service_name]
      regex: (.*)
      target_label: service
      replacement: $1
      action: replace
    # Add the instance label (will contain the Pod name).
    - source_labels: [__meta_kubernetes_pod_name]
      regex: (.*)
      target_label: instance
      replacement: $1
      action: replace
```

Thus, Prometheus automatically tracks the addition and removal of:

- Pods (changes to endpoints lead to the addition or removal of monitoring targets)
- Services (and corresponding endpoints) in the specified namespaces

Configuration changes are required when you need to:

- Add a new `scrape_config` (usually a new class of services to monitor).
- Change the list of namespaces.

Prometheus Operator

Prometheus Operator functions

Using CRDs (Custom Resource Definitions), Prometheus Operator defines four custom resources:

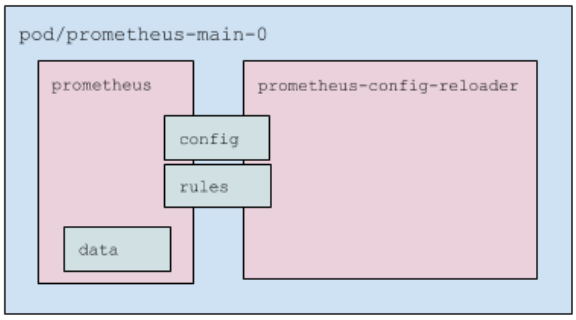
- Prometheus: Describes a Prometheus installation (cluster);
- ServiceMonitor: Specifies how to collect metrics from a set of services;
- Alertmanager: Describes an Alertmanager cluster;
- PrometheusRule: Contains a set of Prometheus rules.

Prometheus Operator also watches Prometheus resources and, for each one, generates:

- StatefulSet with Prometheus itself
- Secret containing `prometheus.yaml` (the main configuration file) and `configmaps.json` (the configuration file for `prometheus-config-reloader`)

It watches ServiceMonitor and PrometheusRule resources and, based on them, updates the configuration (`prometheus.yaml` and `configmaps.json`) in the corresponding Secret.

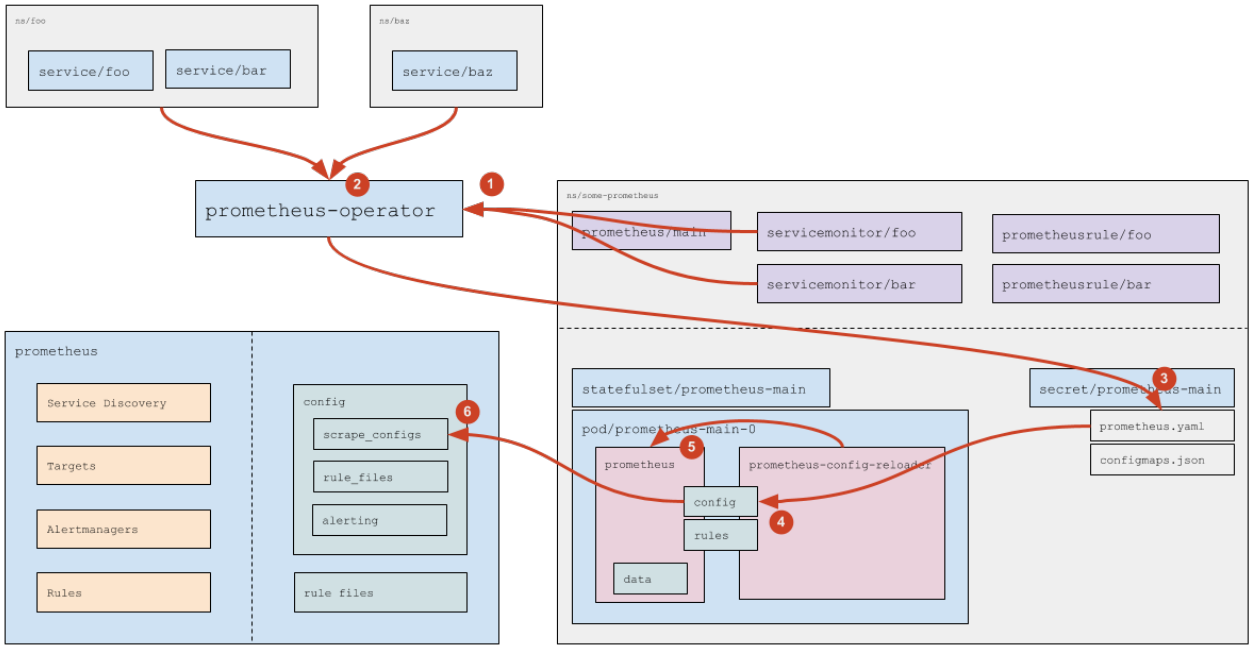
Contents of the Prometheus Pod



Prometheus pods include:

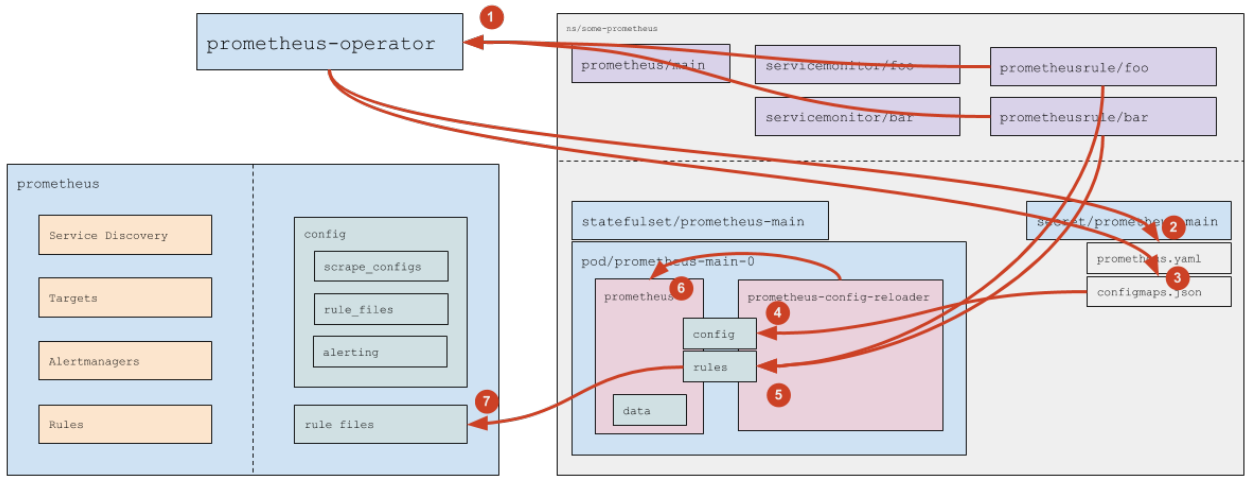
- Three containers:
  - `prometheus`: Prometheus server itself.
  - `prometheus-config-reloader`: Utility that watches:
    - Changes made to `prometheus.yaml` and, if needed, triggers a Prometheus configuration reload (via a special HTTP request; see details below).
    - PrometheusRules (see details below), downloads them if necessary, and restarts Prometheus.
  - `kube-rbac-proxy`: Proxy that handles RBAC-based authentication and authorization for accessing Prometheus metrics.
- Key volumes:
  - `config`: Mounted Secret with the files `prometheus.yaml` and `configmaps.json`. Used by the `prometheus` and `prometheus-config-reloader` containers.
  - `rules`: An `emptyDir` populated by `prometheus-config-reloader` and read by `prometheus`. Mounted into both containers; in `prometheus`, it is read-only.
  - `data`: Prometheus data. Mounted only into the `prometheus` container.

ServiceMonitor processing



1. Prometheus Operator watches ServiceMonitor resources (which ones exactly is defined in the Prometheus resource spec; see details in the official documentation).
2. For each ServiceMonitor that does not specify an explicit list of namespaces (`any: true` is used), the operator queries the Kubernetes API to compute the list of namespaces with Service`s that match the labels from the ServiceMonitor.
3. Based on the discovered ServiceMonitor resources and the computed namespaces, the operator generates the Prometheus configuration fragment (the `scrape_configs` section) and stores it in the corresponding Secret.
4. Kubernetes mounting mechanism updates the files in the Prometheus pod — `prometheus.yaml` from the Secret is refreshed.
5. The `prometheus-config-reloader` container detects the change to `prometheus.yaml` and sends Prometheus an HTTP request to reload the configuration.
6. Prometheus rereads the configuration and applies the changes in `scrape_configs` according to its workflow.

Processing custom resources with rules



1. Prometheus Operator watches PrometheusRule objects that match the `ruleSelector` specified in the Prometheus resource.
2. When a new PrometheusRule appears or an existing one is deleted, the operator updates `prometheus.yaml` — after that, the same chain as for `ServiceMonitor` processing applies.
3. When a PrometheusRule is added, deleted, or its contents change, the operator updates the ConfigMap `prometheus-main-rulefiles-0`.
4. Kubernetes' mounting mechanism delivers the updated ConfigMap to the pod.
5. The `prometheus-config-reloader` container detects the changes and then:
   - writes the updated ConfigMaps to the `rules` directory (`emptyDir`);
   - sends Prometheus an HTTP request to reload the configuration.
6. Prometheus rereads the configuration and applies the updated rules.

Third-party components

List of third-party software used in the `operator-prometheus` module:

- Prometheus Operator

  License: Apache License 2.0

  Kubernetes native deployment and management of Prometheus and related monitoring components.

Configuration

> **i** The module lifecycle stage: General Availability

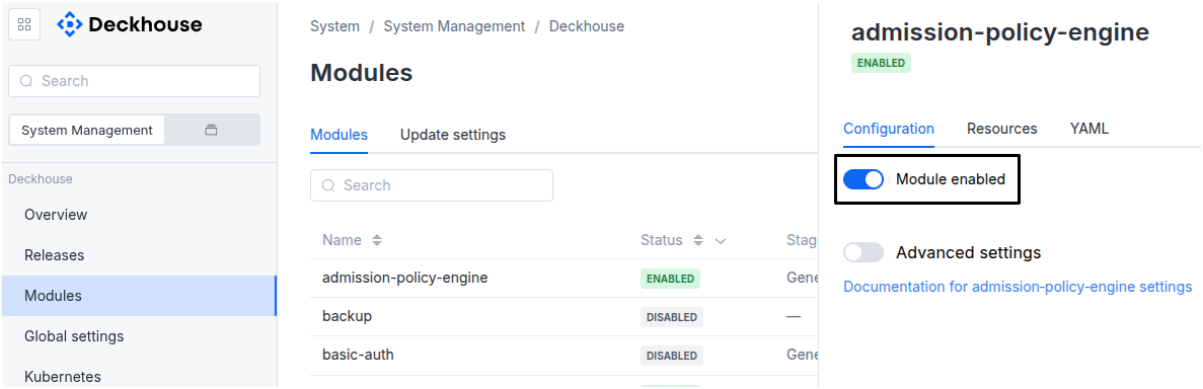
The module has 4 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `operator-prometheus` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable operator-prometheus
```

- **Using ModuleConfig `operator-prometheus`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `operator-prometheus` (create it if necessary);

  Example of a manifest to enable module `operator-prometheus`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: operator-prometheus
spec:
  enabled: true
```

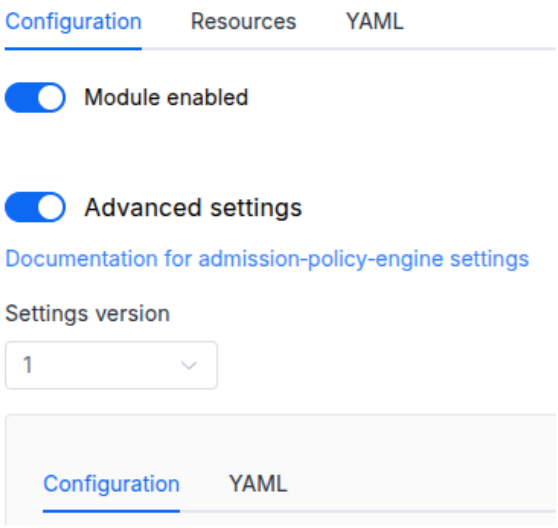
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `operator-prometheus` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `operator-prometheus` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `operator-prometheus`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `operator-prometheus` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc operator-prometheus
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `operator-prometheus` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `operator-prometheus`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: operator-prometheus
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.nodeSelector**
    object

    The same as in the pods’ `spec.nodeSelector` parameter in Kubernetes.

    If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations**
    array of objects

    The same as in the pods’ `spec.tolerations` parameter in Kubernetes;

    If the parameter is omitted or `false`, it will be determined automatically.

- **settings.tolerations.effect**
  string
- **settings.tolerations.key**
  string
- **settings.tolerations.operator**
  string
- **settings.tolerations.tolerationSeconds**
  integer
- **settings.tolerations.value**
  string

Custom resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

PodMonitor

Short names: pmon

Scope: Namespaced
Version: v1

PodMonitor defines monitoring for a set of pods.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

  Required value

  Specification of desired Pod selection for target discovery by Prometheus.

  - **spec.attachMetadata**
    object

    Attaches node metadata to discovered targets. Requires Prometheus v2.35.0 and above.

    - **spec.attachMetadata.node**
      boolean

      When set to true, Prometheus must have permissions to get Nodes.

  - **spec.jobLabel**
    string

    The label to use to retrieve the job name from.

  - **spec.keepDroppedTargets**
    integer

    Per-scrape limit on the number of targets dropped by relabeling that will be kept in memory. 0 means no limit. It requires Prometheus >= v2.47.0.

  - **spec.labelLimit**
    integer

    Per-scrape limit on number of labels that will be accepted for a sample. Only valid in Prometheus

versions 2.27.0 and newer.

- **spec.labelNameLengthLimit**
  integer

  Per-scrape limit on length of labels name that will be accepted for a sample. Only valid in Prometheus versions 2.27.0 and newer.

- **spec.labelValueLengthLimit**
  integer

  Per-scrape limit on length of labels value that will be accepted for a sample. Only valid in Prometheus versions 2.27.0 and newer.

- **spec.namespaceSelector**
  object

  Selector to select which namespaces the Endpoints objects are discovered from.

  - **spec.namespaceSelector.any**
    boolean

    Boolean describing whether all namespaces are selected in contrast to a list restricting them.

  - **spec.namespaceSelector.matchNames**
    array of strings

    List of namespace names to select from.

- **spec.podMetricsEndpoints**
  array of objects

  Required value

  A list of endpoints allowed as part of this PodMonitor.

  PodMetricsEndpoint defines a scrapeable endpoint of a Kubernetes Pod serving Prometheus metrics.

  - **spec.podMetricsEndpoints.authorization**
    object

    Authorization section for this endpoint

    - **spec.podMetricsEndpoints.authorization.credentials**
      object

      Selects a key of a Secret in the namespace that contains the credentials for authentication.

      - **spec.podMetricsEndpoints.authorization.credentials.key**
        string

        Required value

        The key of the secret to select from. Must be a valid secret key.

      - **spec.podMetricsEndpoints.authorization.credentials.name**
        string

        Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

      - **spec.podMetricsEndpoints.authorization.credentials.optional**
        boolean

        Specify whether the Secret or its key must be defined

    - **spec.podMetricsEndpoints.authorization.type**
      string

      Defines the authentication type. The value is case-insensitive. “Basic” is not a supported value. Default: “Bearer”

  - **spec.podMetricsEndpoints.basicAuth**
    object

    BasicAuth allow an endpoint to authenticate over basic authentication. More info: https://prometheus.io/docs/operating/configuration/#endpoint

    - **spec.podMetricsEndpoints.basicAuth.password**
      object

The secret in the service monitor namespace that contains the password for authentication.

- **spec.podMetricsEndpoints.basicAuth.password.key**
  string

  Required value

  The key of the secret to select from. Must be a valid secret key.

- **spec.podMetricsEndpoints.basicAuth.password.name**
  string

  Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.podMetricsEndpoints.basicAuth.password.optional**
  boolean

  Specify whether the Secret or its key must be defined

- **spec.podMetricsEndpoints.basicAuth.username**
  object

  The secret in the service monitor namespace that contains the username for authentication.

  - **spec.podMetricsEndpoints.basicAuth.username.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.podMetricsEndpoints.basicAuth.username.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.podMetricsEndpoints.basicAuth.username.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.podMetricsEndpoints.bearerTokenSecret**
  object

  Secret to mount to read bearer token for scraping targets. The secret needs to be in the same namespace as the pod monitor and accessible by the Prometheus Operator.

  - **spec.podMetricsEndpoints.bearerTokenSecret.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.podMetricsEndpoints.bearerTokenSecret.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.podMetricsEndpoints.bearerTokenSecret.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.podMetricsEndpoints.enableHttp2**
  boolean

  Whether to enable HTTP2.

- **spec.podMetricsEndpoints.filterRunning**
  boolean

  Drop pods that are not running. (Failed, Succeeded). Enabled by default. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#pod-phase

- **spec.podMetricsEndpoints.followRedirects**
  boolean

FollowRedirects configures whether scrape requests follow HTTP 3xx redirects.

- **spec.podMetricsEndpoints.honorLabels**
  boolean

  HonorLabels chooses the metric’s labels on collisions with target labels.

- **spec.podMetricsEndpoints.honorTimestamps**
  boolean

  HonorTimestamps controls whether Prometheus respects the timestamps present in scraped data.

- **spec.podMetricsEndpoints.interval**
  string

  Interval at which metrics should be scraped If not specified Prometheus’ global scrape interval is used.

  Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

- **spec.podMetricsEndpoints.metricRelabelings**
  array of objects

  MetricRelabelConfigs to apply to samples before ingestion.

  RelabelConfig allows dynamic rewriting of the label set for targets, alerts, scraped samples and remote write samples. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

  - **spec.podMetricsEndpoints.metricRelabelings.action**
    string

    Action to perform based on the regex matching. `Uppercase` and `Lowercase` actions require Prometheus >= v2.36.0. `DropEqual` and `KeepEqual` actions require Prometheus >= v2.41.0. Default: “Replace”

    Default: `replace`

    Allowed values: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

  - **spec.podMetricsEndpoints.metricRelabelings.modulus**
    integer

    Modulus to take of the hash of the source label values. Only applicable when the action is `HashMod`.

  - **spec.podMetricsEndpoints.metricRelabelings.regex**
    string

    Regular expression against which the extracted value is matched.

  - **spec.podMetricsEndpoints.metricRelabelings.replacement**
    string

    Replacement value against which a Replace action is performed if the regular expression matches. Regex capture groups are available.

  - **spec.podMetricsEndpoints.metricRelabelings.separator**
    string

    Separator is the string between concatenated SourceLabels.

  - **spec.podMetricsEndpoints.metricRelabelings.sourceLabels**
    array of strings

    The source labels select values from existing labels. Their content is concatenated using the configured Separator and matched against the configured regular expression.

    - **Element of the array**
      string

      LabelName is a valid Prometheus label name which may only contain ASCII letters, numbers, as well as underscores.

      Pattern: `^[a-zA-Z_][a-zA-Z0-9_]*$`

  - **spec.podMetricsEndpoints.metricRelabelings.targetLabel**

string

Label to which the resulting string is written in a replacement. It is mandatory for `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` and `DropEqual` actions. Regex capture groups are available.

- **spec.podMetricsEndpoints.oauth2**
  object

  OAuth2 for the URL. Only valid in Prometheus versions 2.27.0 and newer.

  - **spec.podMetricsEndpoints.oauth2.clientId**
    object

    Required value

    The secret or configmap containing the OAuth2 client id

    - **spec.podMetricsEndpoints.oauth2.clientId.configMap**
      object

      ConfigMap containing data to use for the targets.

      - **spec.podMetricsEndpoints.oauth2.clientId.configMap.key**
        string

        Required value

        The key to select.

      - **spec.podMetricsEndpoints.oauth2.clientId.configMap.name**
        string

        Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

      - **spec.podMetricsEndpoints.oauth2.clientId.configMap.optional**
        boolean

        Specify whether the ConfigMap or its key must be defined

    - **spec.podMetricsEndpoints.oauth2.clientId.secret**
      object

      Secret containing data to use for the targets.

      - **spec.podMetricsEndpoints.oauth2.clientId.secret.key**
        string

        Required value

        The key of the secret to select from. Must be a valid secret key.

      - **spec.podMetricsEndpoints.oauth2.clientId.secret.name**
        string

        Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

      - **spec.podMetricsEndpoints.oauth2.clientId.secret.optional**
        boolean

        Specify whether the Secret or its key must be defined

  - **spec.podMetricsEndpoints.oauth2.clientSecret**
    object

    Required value

    The secret containing the OAuth2 client secret

    - **spec.podMetricsEndpoints.oauth2.clientSecret.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.podMetricsEndpoints.oauth2.clientSecret.name**
      string

Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.podMetricsEndpoints.oauth2.clientSecret.optional**
  boolean

  Specify whether the Secret or its key must be defined

- **spec.podMetricsEndpoints.oauth2.endpointParams**
  object

  Parameters to append to the token URL

- **spec.podMetricsEndpoints.oauth2.scopes**
  array of strings

  OAuth2 scopes used for the token request

- **spec.podMetricsEndpoints.oauth2.tokenUrl**
  string

  Required value

  The URL to fetch the token from

- **spec.podMetricsEndpoints.params**
  object

  Optional HTTP URL parameters

- **spec.podMetricsEndpoints.path**
  string

  HTTP path to scrape for metrics. If empty, Prometheus uses the default value (e.g. `/metrics`).

- **spec.podMetricsEndpoints.port**
  string

  Name of the pod port this endpoint refers to. Mutually exclusive with targetPort.

- **spec.podMetricsEndpoints.proxyUrl**
  string

  ProxyURL eg http://proxyserver:2195 Directs scrapes to proxy through this endpoint.

- **spec.podMetricsEndpoints.relabelings**
  array of objects

  RelabelConfigs to apply to samples before scraping. Prometheus Operator automatically adds relabelings for a few standard Kubernetes fields. The original scrape job's name is available via the `__tmp_prometheus_job_name` label. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

  RelabelConfig allows dynamic rewriting of the label set for targets, alerts, scraped samples and remote write samples. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

  - **spec.podMetricsEndpoints.relabelings.action**
    string

    Action to perform based on the regex matching. `Uppercase` and `Lowercase` actions require Prometheus >= v2.36.0. `DropEqual` and `KeepEqual` actions require Prometheus >= v2.41.0. Default: "Replace"

    Default: `replace`

    Allowed values: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

  - **spec.podMetricsEndpoints.relabelings.modulus**
    integer

    Modulus to take of the hash of the source label values. Only applicable when the action is `HashMod`.

  - **spec.podMetricsEndpoints.relabelings.regex**
    string

    Regular expression against which the extracted value is matched.

- **spec.podMetricsEndpoints.relabelings.replacement**
  string

  Replacement value against which a Replace action is performed if the regular expression matches. Regex capture groups are available.

- **spec.podMetricsEndpoints.relabelings.separator**
  string

  Separator is the string between concatenated SourceLabels.

- **spec.podMetricsEndpoints.relabelings.sourceLabels**
  array of strings

  The source labels select values from existing labels. Their content is concatenated using the configured Separator and matched against the configured regular expression.

  - **Element of the array**
    string

    LabelName is a valid Prometheus label name which may only contain ASCII letters, numbers, as well as underscores.

    Pattern: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.podMetricsEndpoints.relabelings.targetLabel**
  string

  Label to which the resulting string is written in a replacement. It is mandatory for `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` and `DropEqual` actions. Regex capture groups are available.

- **spec.podMetricsEndpoints.scheme**
  string

  HTTP scheme to use for scraping. `http` and `https` are the expected values unless you rewrite the `__scheme__` label via relabeling. If empty, Prometheus uses the default value `http`.

  Allowed values: `http`, `https`

- **spec.podMetricsEndpoints.scrapeTimeout**
  string

  Timeout after which the scrape is ended If not specified, the Prometheus global scrape interval is used.

  Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

- **spec.podMetricsEndpoints.targetPort**
  integer or string

  Deprecated: Use ‘port’ instead.

- **spec.podMetricsEndpoints.tlsConfig**
  object

  TLS configuration to use when scraping the endpoint.

  - **spec.podMetricsEndpoints.tlsConfig.ca**
    object

    Certificate authority used when verifying server certificates.

    - **spec.podMetricsEndpoints.tlsConfig.ca.configMap**
      object

      ConfigMap containing data to use for the targets.

      - **spec.podMetricsEndpoints.tlsConfig.ca.configMap.key**
        string

        Required value

        The key to select.

      - **spec.podMetricsEndpoints.tlsConfig.ca.configMap.name**
        string

        Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

        - **spec.podMetricsEndpoints.tlsConfig.ca.configMap.optional**
          boolean

          Specify whether the ConfigMap or its key must be defined

      - **spec.podMetricsEndpoints.tlsConfig.ca.secret**
        object

        Secret containing data to use for the targets.

        - **spec.podMetricsEndpoints.tlsConfig.ca.secret.key**
          string

          Required value

          The key of the secret to select from. Must be a valid secret key.

        - **spec.podMetricsEndpoints.tlsConfig.ca.secret.name**
          string

          Name of the referent. More info:
          https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

        - **spec.podMetricsEndpoints.tlsConfig.ca.secret.optional**
          boolean

          Specify whether the Secret or its key must be defined

    - **spec.podMetricsEndpoints.tlsConfig.cert**
      object

      Client certificate to present when doing client-authentication.

      - **spec.podMetricsEndpoints.tlsConfig.cert.configMap**
        object

        ConfigMap containing data to use for the targets.

        - **spec.podMetricsEndpoints.tlsConfig.cert.configMap.key**
          string

          Required value

          The key to select.

        - **spec.podMetricsEndpoints.tlsConfig.cert.configMap.name**
          string

          Name of the referent. More info:
          https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

        - **spec.podMetricsEndpoints.tlsConfig.cert.configMap.optional**
          boolean

          Specify whether the ConfigMap or its key must be defined

      - **spec.podMetricsEndpoints.tlsConfig.cert.secret**
        object

        Secret containing data to use for the targets.

        - **spec.podMetricsEndpoints.tlsConfig.cert.secret.key**
          string

          Required value

          The key of the secret to select from. Must be a valid secret key.

        - **spec.podMetricsEndpoints.tlsConfig.cert.secret.name**
          string

          Name of the referent. More info:
          https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

        - **spec.podMetricsEndpoints.tlsConfig.cert.secret.optional**
          boolean

          Specify whether the Secret or its key must be defined

    - **spec.podMetricsEndpoints.tlsConfig.insecureSkipVerify**

boolean

Disable target certificate validation.

- **spec.podMetricsEndpoints.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

  - **spec.podMetricsEndpoints.tlsConfig.keySecret.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.podMetricsEndpoints.tlsConfig.keySecret.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.podMetricsEndpoints.tlsConfig.keySecret.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.podMetricsEndpoints.tlsConfig.serverName**
  string

  Used to verify the hostname for the targets.

- **spec.podTargetLabels**
  array of strings

  PodTargetLabels transfers labels on the Kubernetes Pod onto the target.

- **spec.sampleLimit**
  integer

  SampleLimit defines per-scrape limit on number of scraped samples that will be accepted.

- **spec.selector**
  object

  Required value

  Selector to select Pod objects.

  - **spec.selector.matchExpressions**
    array of objects

    matchExpressions is a list of label selector requirements. The requirements are ANDed.

    A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

    - **spec.selector.matchExpressions.key**
      string

      Required value

      key is the label key that the selector applies to.

    - **spec.selector.matchExpressions.operator**
      string

      Required value

      operator represents a key’s relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

    - **spec.selector.matchExpressions.values**
      array of strings

      values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

  - **spec.selector.matchLabels**
    object

    matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is

equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are ANDed.

- **spec.targetLimit**
  integer

  TargetLimit defines a limit on the number of scraped targets that will be accepted.

Probe

Short names: prb

Scope: Namespaced
Version: v1

Probe defines monitoring for a set of static targets or ingresses.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

  Required value

  Specification of desired Ingress selection for target discovery by Prometheus.

  - **spec.authorization**
    object

    Authorization section for this endpoint

    - **spec.authorization.credentials**
      object

      Selects a key of a Secret in the namespace that contains the credentials for authentication.

      - **spec.authorization.credentials.key**
        string

        Required value

        The key of the secret to select from. Must be a valid secret key.

      - **spec.authorization.credentials.name**
        string

        Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

      - **spec.authorization.credentials.optional**
        boolean

        Specify whether the Secret or its key must be defined

    - **spec.authorization.type**
      string

      Defines the authentication type. The value is case-insensitive. “Basic” is not a supported value. Default: “Bearer”

  - **spec.basicAuth**
    object

    BasicAuth allow an endpoint to authenticate over basic authentication. More info: https://prometheus.io/docs/operating/configuration/#endpoint

    - **spec.basicAuth.password**
      object

The secret in the service monitor namespace that contains the password for authentication.

- **spec.basicAuth.password.key**
  string

  Required value

  The key of the secret to select from. Must be a valid secret key.

- **spec.basicAuth.password.name**
  string

  Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.basicAuth.password.optional**
  boolean

  Specify whether the Secret or its key must be defined

- **spec.basicAuth.username**
  object

  The secret in the service monitor namespace that contains the username for authentication.

  - **spec.basicAuth.username.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.basicAuth.username.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.basicAuth.username.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.bearerTokenSecret**
  object

  Secret to mount to read bearer token for scraping targets. The secret needs to be in the same namespace as the probe and accessible by the Prometheus Operator.

  - **spec.bearerTokenSecret.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.bearerTokenSecret.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.bearerTokenSecret.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.interval**
  string

  Interval at which targets are probed using the configured prober. If not specified Prometheus' global scrape interval is used.

  Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

- **spec.jobName**
  string

  The job name assigned to scraped metrics by default.

- **spec.keepDroppedTargets**
  integer

Per-scrape limit on the number of targets dropped by relabeling that will be kept in memory. 0 means no limit. It requires Prometheus >= v2.47.0.

- **spec.labelLimit**
  integer

  Per-scrape limit on number of labels that will be accepted for a sample. Only valid in Prometheus versions 2.27.0 and newer.

- **spec.labelNameLengthLimit**
  integer

  Per-scrape limit on length of labels name that will be accepted for a sample. Only valid in Prometheus versions 2.27.0 and newer.

- **spec.labelValueLengthLimit**
  integer

  Per-scrape limit on length of labels value that will be accepted for a sample. Only valid in Prometheus versions 2.27.0 and newer.

- **spec.metricRelabelings**
  array of objects

  MetricRelabelConfigs to apply to samples before ingestion.

  RelabelConfig allows dynamic rewriting of the label set for targets, alerts, scraped samples and remote write samples. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

  - **spec.metricRelabelings.action**
    string

    Action to perform based on the regex matching. `Uppercase` and `Lowercase` actions require Prometheus >= v2.36.0. `DropEqual` and `KeepEqual` actions require Prometheus >= v2.41.0. Default: “Replace”

    Default: `replace`

    Allowed values: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

  - **spec.metricRelabelings.modulus**
    integer

    Modulus to take of the hash of the source label values. Only applicable when the action is `HashMod`.

  - **spec.metricRelabelings.regex**
    string

    Regular expression against which the extracted value is matched.

  - **spec.metricRelabelings.replacement**
    string

    Replacement value against which a Replace action is performed if the regular expression matches. Regex capture groups are available.

  - **spec.metricRelabelings.separator**
    string

    Separator is the string between concatenated SourceLabels.

  - **spec.metricRelabelings.sourceLabels**
    array of strings

    The source labels select values from existing labels. Their content is concatenated using the configured Separator and matched against the configured regular expression.

    - **Element of the array**
      string

      LabelName is a valid Prometheus label name which may only contain ASCII letters, numbers, as well as underscores.

      Pattern: `^[a-zA-Z_][a-zA-Z0-9_]*$`

  - **spec.metricRelabelings.targetLabel**
    string

Label to which the resulting string is written in a replacement. It is mandatory for `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` and `DropEqual` actions. Regex capture groups are available.

- **spec.module**
  string

  The module to use for probing specifying how to probe the target. Example module configuring in the blackbox exporter:
  https://github.com/prometheus/blackbox_exporter/blob/master/example.yml

- **spec.oauth2**
  object

  OAuth2 for the URL. Only valid in Prometheus versions 2.27.0 and newer.

  - **spec.oauth2.clientId**
    object

    Required value

    The secret or configmap containing the OAuth2 client id

    - **spec.oauth2.clientId.configMap**
      object

      ConfigMap containing data to use for the targets.

      - **spec.oauth2.clientId.configMap.key**
        string

        Required value

        The key to select.

      - **spec.oauth2.clientId.configMap.name**
        string

        Name of the referent. More info:
        https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

      - **spec.oauth2.clientId.configMap.optional**
        boolean

        Specify whether the ConfigMap or its key must be defined

    - **spec.oauth2.clientId.secret**
      object

      Secret containing data to use for the targets.

      - **spec.oauth2.clientId.secret.key**
        string

        Required value

        The key of the secret to select from. Must be a valid secret key.

      - **spec.oauth2.clientId.secret.name**
        string

        Name of the referent. More info:
        https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

      - **spec.oauth2.clientId.secret.optional**
        boolean

        Specify whether the Secret or its key must be defined

  - **spec.oauth2.clientSecret**
    object

    Required value

    The secret containing the OAuth2 client secret

    - **spec.oauth2.clientSecret.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

        - **spec.oauth2.clientSecret.name**
          string

          Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

        - **spec.oauth2.clientSecret.optional**
          boolean

          Specify whether the Secret or its key must be defined

      - **spec.oauth2.endpointParams**
        object

        Parameters to append to the token URL

      - **spec.oauth2.scopes**
        array of strings

        OAuth2 scopes used for the token request

      - **spec.oauth2.tokenUrl**
        string

        Required value

        The URL to fetch the token from

  - **spec.prober**
    object

    Specification for the prober to use for probing targets. The prober.URL parameter is required. Targets cannot be probed if left empty.

      - **spec.prober.path**
        string

        Path to collect metrics from. Defaults to `/probe`.

        Default: `/probe`

      - **spec.prober.proxyUrl**
        string

        Optional ProxyURL.

      - **spec.prober.scheme**
        string

        HTTP scheme to use for scraping. `http` and `https` are the expected values unless you rewrite the `__scheme__` label via relabeling. If empty, Prometheus uses the default value `http`.

        Allowed values: `http`, `https`

      - **spec.prober.url**
        string

        Required value

        Mandatory URL of the prober.

  - **spec.sampleLimit**
    integer

    SampleLimit defines per-scrape limit on number of scraped samples that will be accepted.

  - **spec.scrapeTimeout**
    string

    Timeout for scraping metrics from the Prometheus exporter. If not specified, the Prometheus global scrape timeout is used.

    Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

  - **spec.targetLimit**
    integer

    TargetLimit defines a limit on the number of scraped targets that will be accepted.

  - **spec.targets**
    object

Targets defines a set of static or dynamically discovered targets to probe.

- **spec.targets.ingress**
  object

  ingress defines the Ingress objects to probe and the relabeling configuration. If `staticConfig` is also defined, `staticConfig` takes precedence.

  - **spec.targets.ingress.namespaceSelector**
    object

    From which namespaces to select Ingress objects.

    - **spec.targets.ingress.namespaceSelector.any**
      boolean

      Boolean describing whether all namespaces are selected in contrast to a list restricting them.

    - **spec.targets.ingress.namespaceSelector.matchNames**
      array of strings

      List of namespace names to select from.

  - **spec.targets.ingress.relabelingConfigs**
    array of objects

    RelabelConfigs to apply to the label set of the target before it gets scraped. The original ingress address is available via the `__tmp_prometheus_ingress_address` label. It can be used to customize the probed URL. The original scrape job's name is available via the `__tmp_prometheus_job_name` label. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

    RelabelConfig allows dynamic rewriting of the label set for targets, alerts, scraped samples and remote write samples. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

    - **spec.targets.ingress.relabelingConfigs.action**
      string

      Action to perform based on the regex matching. `Uppercase` and `Lowercase` actions require Prometheus >= v2.36.0. `DropEqual` and `KeepEqual` actions require Prometheus >= v2.41.0. Default: "Replace"

      Default: `replace`

      Allowed values: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

    - **spec.targets.ingress.relabelingConfigs.modulus**
      integer

      Modulus to take of the hash of the source label values. Only applicable when the action is `HashMod`.

    - **spec.targets.ingress.relabelingConfigs.regex**
      string

      Regular expression against which the extracted value is matched.

    - **spec.targets.ingress.relabelingConfigs.replacement**
      string

      Replacement value against which a Replace action is performed if the regular expression matches. Regex capture groups are available.

    - **spec.targets.ingress.relabelingConfigs.separator**
      string

      Separator is the string between concatenated SourceLabels.

    - **spec.targets.ingress.relabelingConfigs.sourceLabels**
      array of strings

      The source labels select values from existing labels. Their content is concatenated using the configured Separator and matched against the configured regular expression.

      - **Element of the array**
        string

LabelName is a valid Prometheus label name which may only contain ASCII letters, numbers, as well as underscores.

Pattern: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.targets.ingress.relabelingConfigs.targetLabel**
  string

  Label to which the resulting string is written in a replacement. It is mandatory for `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` and `DropEqual` actions. Regex capture groups are available.

- **spec.targets.ingress.selector**
  object

  Selector to select the Ingress objects.

  - **spec.targets.ingress.selector.matchExpressions**
    array of objects

    matchExpressions is a list of label selector requirements. The requirements are ANDed.

    A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

    - **spec.targets.ingress.selector.matchExpressions.key**
      string

      Required value

      key is the label key that the selector applies to.

    - **spec.targets.ingress.selector.matchExpressions.operator**
      string

      Required value

      operator represents a key’s relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

    - **spec.targets.ingress.selector.matchExpressions.values**
      array of strings

      values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

  - **spec.targets.ingress.selector.matchLabels**
    object

    matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is “key”, the operator is “In”, and the values array contains only “value”. The requirements are ANDed.

- **spec.targets.staticConfig**
  object

  staticConfig defines the static list of targets to probe and the relabeling configuration. If `ingress` is also defined, `staticConfig` takes precedence. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#static_config.

  - **spec.targets.staticConfig.labels**
    object

    Labels assigned to all metrics scraped from the targets.

  - **spec.targets.staticConfig.relabelingConfigs**
    array of objects

    RelabelConfigs to apply to the label set of the targets before it gets scraped. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

    RelabelConfig allows dynamic rewriting of the label set for targets, alerts, scraped samples and remote write samples. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

    - **spec.targets.staticConfig.relabelingConfigs.action**
      string

Action to perform based on the regex matching. `Uppercase` and `Lowercase` actions require Prometheus >= v2.36.0. `DropEqual` and `KeepEqual` actions require Prometheus >= v2.41.0. Default: “Replace”

Default: `replace`

Allowed values: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

- **spec.targets.staticConfig.relabelingConfigs.modulus**
  integer

  Modulus to take of the hash of the source label values. Only applicable when the action is `HashMod`.

- **spec.targets.staticConfig.relabelingConfigs.regex**
  string

  Regular expression against which the extracted value is matched.

- **spec.targets.staticConfig.relabelingConfigs.replacement**
  string

  Replacement value against which a Replace action is performed if the regular expression matches. Regex capture groups are available.

- **spec.targets.staticConfig.relabelingConfigs.separator**
  string

  Separator is the string between concatenated SourceLabels.

- **spec.targets.staticConfig.relabelingConfigs.sourceLabels**
  array of strings

  The source labels select values from existing labels. Their content is concatenated using the configured Separator and matched against the configured regular expression.

  - **Element of the array**
    string

    LabelName is a valid Prometheus label name which may only contain ASCII letters, numbers, as well as underscores.

    Pattern: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.targets.staticConfig.relabelingConfigs.targetLabel**
  string

  Label to which the resulting string is written in a replacement. It is mandatory for `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` and `DropEqual` actions. Regex capture groups are available.

- **spec.targets.staticConfig.static**
  array of strings

  The list of hosts to probe.

- **spec.tlsConfig**
  object

  TLS configuration to use when scraping the endpoint.

  - **spec.tlsConfig.ca**
    object

    Certificate authority used when verifying server certificates.

    - **spec.tlsConfig.ca.configMap**
      object

      ConfigMap containing data to use for the targets.

      - **spec.tlsConfig.ca.configMap.key**
        string

        Required value

        The key to select.

- **spec.tlsConfig.ca.configMap.name**
  string

  Name of the referent. More info:
  https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.tlsConfig.ca.configMap.optional**
  boolean

  Specify whether the ConfigMap or its key must be defined

- **spec.tlsConfig.ca.secret**
  object

  Secret containing data to use for the targets.

  - **spec.tlsConfig.ca.secret.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.tlsConfig.ca.secret.name**
    string

    Name of the referent. More info:
    https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.tlsConfig.ca.secret.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.tlsConfig.cert**
  object

  Client certificate to present when doing client-authentication.

  - **spec.tlsConfig.cert.configMap**
    object

    ConfigMap containing data to use for the targets.

    - **spec.tlsConfig.cert.configMap.key**
      string

      Required value

      The key to select.

    - **spec.tlsConfig.cert.configMap.name**
      string

      Name of the referent. More info:
      https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.tlsConfig.cert.configMap.optional**
      boolean

      Specify whether the ConfigMap or its key must be defined

  - **spec.tlsConfig.cert.secret**
    object

    Secret containing data to use for the targets.

    - **spec.tlsConfig.cert.secret.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.tlsConfig.cert.secret.name**
      string

      Name of the referent. More info:
      https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.tlsConfig.cert.secret.optional**
      boolean

Specify whether the Secret or its key must be defined

- **spec.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

  - **spec.tlsConfig.keySecret.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.tlsConfig.keySecret.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.tlsConfig.keySecret.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.tlsConfig.serverName**
  string

  Used to verify the hostname for the targets.

ScrapeConfig

Short names: scfg

Scope: Namespaced
Version: v1alpha1

ScrapeConfig defines a namespaced Prometheus scrape_config to be aggregated across multiple namespaces into the Prometheus configuration.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

  Required value

  ScrapeConfigSpec is a specification of the desired configuration for a scrape configuration.

  - **spec.authorization**
    object

    Authorization header to use on every scrape request.

    - **spec.authorization.credentials**
      object

      Selects a key of a Secret in the namespace that contains the credentials for authentication.

      - **spec.authorization.credentials.key**
        string

        Required value

The key of the secret to select from. Must be a valid secret key.

- **spec.authorization.credentials.name**
  string

  Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.authorization.credentials.optional**
  boolean

  Specify whether the Secret or its key must be defined

- **spec.authorization.type**
  string

  Defines the authentication type. The value is case-insensitive. “Basic” is not a supported value. Default: “Bearer”

- **spec.basicAuth**
  object

  BasicAuth information to use on every scrape request.

  - **spec.basicAuth.password**
    object

    The secret in the service monitor namespace that contains the password for authentication.

    - **spec.basicAuth.password.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.basicAuth.password.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.basicAuth.password.optional**
      boolean

      Specify whether the Secret or its key must be defined

  - **spec.basicAuth.username**
    object

    The secret in the service monitor namespace that contains the username for authentication.

    - **spec.basicAuth.username.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.basicAuth.username.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.basicAuth.username.optional**
      boolean

      Specify whether the Secret or its key must be defined

- **spec.consulSDConfigs**
  array of objects

  ConsulSDConfigs defines a list of Consul service discovery configurations.

  ConsulSDConfig defines a Consul service discovery configuration See https://prometheus.io/docs/prometheus/latest/configuration/configuration/#consul_sd_config

  - **spec.consulSDConfigs.allow_stale**
    boolean

    Allow stale Consul results (see https://www.consul.io/api/features/consistency.html). Will reduce load on Consul. If unset, Prometheus uses its default value.

- **spec.consulSDConfigs.authorization**
  object

  Authorization header configuration to authenticate against the Consul Server.

  - **spec.consulSDConfigs.authorization.credentials**
    object

    Selects a key of a Secret in the namespace that contains the credentials for authentication.

    - **spec.consulSDConfigs.authorization.credentials.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.consulSDConfigs.authorization.credentials.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.consulSDConfigs.authorization.credentials.optional**
      boolean

      Specify whether the Secret or its key must be defined

  - **spec.consulSDConfigs.authorization.type**
    string

    Defines the authentication type. The value is case-insensitive. “Basic” is not a supported value. Default: “Bearer”

- **spec.consulSDConfigs.basicAuth**
  object

  BasicAuth information to authenticate against the Consul Server. More info: https://prometheus.io/docs/operating/configuration/#endpoints

  - **spec.consulSDConfigs.basicAuth.password**
    object

    The secret in the service monitor namespace that contains the password for authentication.

    - **spec.consulSDConfigs.basicAuth.password.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.consulSDConfigs.basicAuth.password.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.consulSDConfigs.basicAuth.password.optional**
      boolean

      Specify whether the Secret or its key must be defined

  - **spec.consulSDConfigs.basicAuth.username**
    object

    The secret in the service monitor namespace that contains the username for authentication.

    - **spec.consulSDConfigs.basicAuth.username.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.consulSDConfigs.basicAuth.username.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

        - **spec.consulSDConfigs.basicAuth.username.optional**
          boolean

          Specify whether the Secret or its key must be defined

- **spec.consulSDConfigs.datacenter**
  string

  Consul Datacenter name, if not provided it will use the local Consul Agent Datacenter.

- **spec.consulSDConfigs.enable_http2**
  boolean

  Whether to enable HTTP2. If unset, Prometheus uses its default value.

- **spec.consulSDConfigs.follow_redirects**
  boolean

  Configure whether HTTP requests follow HTTP 3xx redirects. If unset, Prometheus uses its default value.

- **spec.consulSDConfigs.namespace**
  string

  Namespaces are only supported in Consul Enterprise.

- **spec.consulSDConfigs.no_proxy**
  string

  Comma-separated string that can contain IPs, CIDR notation, domain names that should be excluded from proxying. IP and domain names can contain port numbers.

- **spec.consulSDConfigs.node_meta**
  object

  Node metadata key/value pairs to filter nodes for a given service.

- **spec.consulSDConfigs.oauth2**
  object

  Optional OAuth 2.0 configuration.

    - **spec.consulSDConfigs.oauth2.clientId**
      object

      Required value

      The secret or configmap containing the OAuth2 client id

        - **spec.consulSDConfigs.oauth2.clientId.configMap**
          object

          ConfigMap containing data to use for the targets.

            - **spec.consulSDConfigs.oauth2.clientId.configMap.key**
              string

              Required value

              The key to select.

            - **spec.consulSDConfigs.oauth2.clientId.configMap.name**
              string

              Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

            - **spec.consulSDConfigs.oauth2.clientId.configMap.optional**
              boolean

              Specify whether the ConfigMap or its key must be defined

        - **spec.consulSDConfigs.oauth2.clientId.secret**
          object

          Secret containing data to use for the targets.

            - **spec.consulSDConfigs.oauth2.clientId.secret.key**
              string

              Required value

The key of the secret to select from. Must be a valid secret key.

- **spec.consulSDConfigs.oauth2.clientId.secret.name**
  string

  Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.consulSDConfigs.oauth2.clientId.secret.optional**
  boolean

  Specify whether the Secret or its key must be defined

- **spec.consulSDConfigs.oauth2.clientSecret**
  object

  Required value

  The secret containing the OAuth2 client secret

  - **spec.consulSDConfigs.oauth2.clientSecret.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.consulSDConfigs.oauth2.clientSecret.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.consulSDConfigs.oauth2.clientSecret.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.consulSDConfigs.oauth2.endpointParams**
  object

  Parameters to append to the token URL

- **spec.consulSDConfigs.oauth2.scopes**
  array of strings

  OAuth2 scopes used for the token request

- **spec.consulSDConfigs.oauth2.tokenUrl**
  string

  Required value

  The URL to fetch the token from

- **spec.consulSDConfigs.partition**
  string

  Admin Partitions are only supported in Consul Enterprise.

- **spec.consulSDConfigs.proxy_connect_header**
  object

  Specifies headers to send to proxies during CONNECT requests.

  - **spec.consulSDConfigs.proxy_connect_header.<KEY_NAME>**
    object

    `<KEY_NAME>` — item (key) name.

    SecretKeySelector selects a key of a Secret.

    - **spec.consulSDConfigs.proxy_connect_header.<KEY_NAME>.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.consulSDConfigs.proxy_connect_header.<KEY_NAME>.name**
      string

Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.consulSDConfigs.proxy_connect_header.<KEY_NAME>.optional**
  boolean

  Specify whether the Secret or its key must be defined

- **spec.consulSDConfigs.proxy_from_environment**
  boolean

  Use proxy URL indicated by environment variables (HTTP_PROXY, https_proxy, HTTPs_PROXY, https_proxy, and no_proxy) If unset, Prometheus uses its default value.

- **spec.consulSDConfigs.proxy_url**
  string

  Optional proxy URL.

- **spec.consulSDConfigs.refresh_interval**
  string

  The time after which the provided names are refreshed. On large setup it might be a good idea to increase this value because the catalog will change all the time. If unset, Prometheus uses its default value.

  Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

- **spec.consulSDConfigs.scheme**
  string

  HTTP Scheme default “http”

  Allowed values: `HTTP`, `HTTPS`

- **spec.consulSDConfigs.server**
  string

  Required value

  A valid string consisting of a hostname or IP followed by an optional port number.

- **spec.consulSDConfigs.services**
  array of strings

  A list of services for which targets are retrieved. If omitted, all services are scraped.

- **spec.consulSDConfigs.tag_separator**
  string

  The string by which Consul tags are joined into the tag label. If unset, Prometheus uses its default value.

- **spec.consulSDConfigs.tags**
  array of strings

  An optional list of tags used to filter nodes for a given service. Services must contain all tags in the list.

- **spec.consulSDConfigs.tlsConfig**
  object

  TLS Config

  - **spec.consulSDConfigs.tlsConfig.ca**
    object

    Certificate authority used when verifying server certificates.

    - **spec.consulSDConfigs.tlsConfig.ca.configMap**
      object

      ConfigMap containing data to use for the targets.

      - **spec.consulSDConfigs.tlsConfig.ca.configMap.key**
        string

        Required value

        The key to select.

      - **spec.consulSDConfigs.tlsConfig.ca.configMap.name**

string

Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.consulSDConfigs.tlsConfig.ca.configMap.optional**
boolean

Specify whether the ConfigMap or its key must be defined

- **spec.consulSDConfigs.tlsConfig.ca.secret**
object

Secret containing data to use for the targets.

- **spec.consulSDConfigs.tlsConfig.ca.secret.key**
string

Required value

The key of the secret to select from. Must be a valid secret key.

- **spec.consulSDConfigs.tlsConfig.ca.secret.name**
string

Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.consulSDConfigs.tlsConfig.ca.secret.optional**
boolean

Specify whether the Secret or its key must be defined

- **spec.consulSDConfigs.tlsConfig.cert**
object

Client certificate to present when doing client-authentication.

- **spec.consulSDConfigs.tlsConfig.cert.configMap**
object

ConfigMap containing data to use for the targets.

- **spec.consulSDConfigs.tlsConfig.cert.configMap.key**
string

Required value

The key to select.

- **spec.consulSDConfigs.tlsConfig.cert.configMap.name**
string

Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.consulSDConfigs.tlsConfig.cert.configMap.optional**
boolean

Specify whether the ConfigMap or its key must be defined

- **spec.consulSDConfigs.tlsConfig.cert.secret**
object

Secret containing data to use for the targets.

- **spec.consulSDConfigs.tlsConfig.cert.secret.key**
string

Required value

The key of the secret to select from. Must be a valid secret key.

- **spec.consulSDConfigs.tlsConfig.cert.secret.name**
string

Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.consulSDConfigs.tlsConfig.cert.secret.optional**
  boolean

  Specify whether the Secret or its key must be defined

- **spec.consulSDConfigs.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.consulSDConfigs.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

  - **spec.consulSDConfigs.tlsConfig.keySecret.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.consulSDConfigs.tlsConfig.keySecret.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.consulSDConfigs.tlsConfig.keySecret.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.consulSDConfigs.tlsConfig.serverName**
  string

  Used to verify the hostname for the targets.

- **spec.consulSDConfigs.tokenRef**
  object

  Consul ACL TokenRef, if not provided it will use the ACL from the local Consul Agent.

  - **spec.consulSDConfigs.tokenRef.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.consulSDConfigs.tokenRef.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.consulSDConfigs.tokenRef.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.dnsSDConfigs**
  array of objects

  DNSSDConfigs defines a list of DNS service discovery configurations.

  DNSSDConfig allows specifying a set of DNS domain names which are periodically queried to discover a list of targets. The DNS servers to be contacted are read from /etc/resolv.conf. See https://prometheus.io/docs/prometheus/latest/configuration/configuration/#dns_sd_config

  - **spec.dnsSDConfigs.names**
    array of strings

    Required value

    A list of DNS domain names to be queried.

  - **spec.dnsSDConfigs.port**
    integer

    The port number used if the query type is not SRV Ignored for SRV records

  - **spec.dnsSDConfigs.refreshInterval**

string

RefreshInterval configures the time after which the provided names are refreshed. If not set, Prometheus uses its default value.

Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

- **spec.dnsSDConfigs.type**
  string

  The type of DNS query to perform. One of SRV, A, AAAA or MX. If not set, Prometheus uses its default value.

  Allowed values: `SRV`, `A`, `AAAA`, `MX`

- **spec.fileSDConfigs**
  array of objects

  FileSDConfigs defines a list of file service discovery configurations.

  FileSDConfig defines a Prometheus file service discovery configuration See https://prometheus.io/docs/prometheus/latest/configuration/configuration/#file_sd_config

  - **spec.fileSDConfigs.files**
    array of strings

    Required value

    List of files to be used for file discovery. Recommendation: use absolute paths. While relative paths work, the prometheus-operator project makes no guarantees about the working directory where the configuration file is stored. Files must be mounted using Prometheus.ConfigMaps or Prometheus.Secrets.

    - **Element of the array**
      string

      SDFile represents a file used for service discovery

      Pattern: `^[^*]*(\*[^/]*)?\.(json|yml|yaml|JSON|YML|YAML)$`

  - **spec.fileSDConfigs.refreshInterval**
    string

    RefreshInterval configures the refresh interval at which Prometheus will reload the content of the files.

    Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

- **spec.honorLabels**
  boolean

  HonorLabels chooses the metric’s labels on collisions with target labels.

- **spec.honorTimestamps**
  boolean

  HonorTimestamps controls whether Prometheus respects the timestamps present in scraped data.

- **spec.httpSDConfigs**
  array of objects

  HTTPSDConfigs defines a list of HTTP service discovery configurations.

  HTTPSDConfig defines a prometheus HTTP service discovery configuration See https://prometheus.io/docs/prometheus/latest/configuration/configuration/#http_sd_config

  - **spec.httpSDConfigs.authorization**
    object

    Authorization header configuration to authenticate against the target HTTP endpoint.

    - **spec.httpSDConfigs.authorization.credentials**
      object

      Selects a key of a Secret in the namespace that contains the credentials for authentication.

      - **spec.httpSDConfigs.authorization.credentials.key**
        string

        Required value

The key of the secret to select from. Must be a valid secret key.

- **spec.httpSDConfigs.authorization.credentials.name**
  string

  Name of the referent. More info:
  https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.httpSDConfigs.authorization.credentials.optional**
  boolean

  Specify whether the Secret or its key must be defined

- **spec.httpSDConfigs.authorization.type**
  string

  Defines the authentication type. The value is case-insensitive. “Basic” is not a supported value. Default: “Bearer”

- **spec.httpSDConfigs.basicAuth**
  object

  BasicAuth information to authenticate against the target HTTP endpoint. More info:
  https://prometheus.io/docs/operating/configuration/#endpoints

  - **spec.httpSDConfigs.basicAuth.password**
    object

    The secret in the service monitor namespace that contains the password for authentication.

    - **spec.httpSDConfigs.basicAuth.password.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.httpSDConfigs.basicAuth.password.name**
      string

      Name of the referent. More info:
      https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.httpSDConfigs.basicAuth.password.optional**
      boolean

      Specify whether the Secret or its key must be defined

  - **spec.httpSDConfigs.basicAuth.username**
    object

    The secret in the service monitor namespace that contains the username for authentication.

    - **spec.httpSDConfigs.basicAuth.username.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.httpSDConfigs.basicAuth.username.name**
      string

      Name of the referent. More info:
      https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.httpSDConfigs.basicAuth.username.optional**
      boolean

      Specify whether the Secret or its key must be defined

- **spec.httpSDConfigs.refreshInterval**
  string

  RefreshInterval configures the refresh interval at which Prometheus will re-query the endpoint to update the target list.

  Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

- **spec.httpSDConfigs.tlsConfig**

object

TLS configuration applying to the target HTTP endpoint.

- **spec.httpSDConfigs.tlsConfig.ca**
  object

  Certificate authority used when verifying server certificates.

  - **spec.httpSDConfigs.tlsConfig.ca.configMap**
    object

    ConfigMap containing data to use for the targets.

    - **spec.httpSDConfigs.tlsConfig.ca.configMap.key**
      string

      Required value

      The key to select.

    - **spec.httpSDConfigs.tlsConfig.ca.configMap.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.httpSDConfigs.tlsConfig.ca.configMap.optional**
      boolean

      Specify whether the ConfigMap or its key must be defined

  - **spec.httpSDConfigs.tlsConfig.ca.secret**
    object

    Secret containing data to use for the targets.

    - **spec.httpSDConfigs.tlsConfig.ca.secret.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.httpSDConfigs.tlsConfig.ca.secret.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.httpSDConfigs.tlsConfig.ca.secret.optional**
      boolean

      Specify whether the Secret or its key must be defined

- **spec.httpSDConfigs.tlsConfig.cert**
  object

  Client certificate to present when doing client-authentication.

  - **spec.httpSDConfigs.tlsConfig.cert.configMap**
    object

    ConfigMap containing data to use for the targets.

    - **spec.httpSDConfigs.tlsConfig.cert.configMap.key**
      string

      Required value

      The key to select.

    - **spec.httpSDConfigs.tlsConfig.cert.configMap.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.httpSDConfigs.tlsConfig.cert.configMap.optional**
      boolean

Specify whether the ConfigMap or its key must be defined

- **spec.httpSDConfigs.tlsConfig.cert.secret**
  object

  Secret containing data to use for the targets.

  - **spec.httpSDConfigs.tlsConfig.cert.secret.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.httpSDConfigs.tlsConfig.cert.secret.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.httpSDConfigs.tlsConfig.cert.secret.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.httpSDConfigs.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.httpSDConfigs.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

  - **spec.httpSDConfigs.tlsConfig.keySecret.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.httpSDConfigs.tlsConfig.keySecret.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.httpSDConfigs.tlsConfig.keySecret.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.httpSDConfigs.tlsConfig.serverName**
  string

  Used to verify the hostname for the targets.

- **spec.httpSDConfigs.url**
  string

  Required value

  URL from which the targets are fetched.

  Pattern: `^http(s)?://.+$`

- **spec.keepDroppedTargets**
  integer

  Per-scrape limit on the number of targets dropped by relabeling that will be kept in memory. 0 means no limit. It requires Prometheus >= v2.47.0.

- **spec.kubernetesSDConfigs**
  array of objects

  KubernetesSDConfigs defines a list of Kubernetes service discovery configurations.

  KubernetesSDConfig allows retrieving scrape targets from Kubernetes' REST API. See https://prometheus.io/docs/prometheus/latest/configuration/configuration/#kubernetes_sd_config

  - **spec.kubernetesSDConfigs.role**

string

Required value

Role of the Kubernetes entities that should be discovered. Currently the only supported role is “Node”.

Allowed values: `Node`

- **spec.labelLimit**
  integer

  Per-scrape limit on number of labels that will be accepted for a sample. Only valid in Prometheus versions 2.27.0 and newer.

- **spec.labelNameLengthLimit**
  integer

  Per-scrape limit on length of labels name that will be accepted for a sample. Only valid in Prometheus versions 2.27.0 and newer.

- **spec.labelValueLengthLimit**
  integer

  Per-scrape limit on length of labels value that will be accepted for a sample. Only valid in Prometheus versions 2.27.0 and newer.

- **spec.metricRelabelings**
  array of objects

  MetricRelabelConfigs to apply to samples before ingestion.

  RelabelConfig allows dynamic rewriting of the label set for targets, alerts, scraped samples and remote write samples. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

  - **spec.metricRelabelings.action**
    string

    Action to perform based on the regex matching. `Uppercase` and `Lowercase` actions require Prometheus >= v2.36.0. `DropEqual` and `KeepEqual` actions require Prometheus >= v2.41.0. Default: “Replace”

    Default: `replace`

    Allowed values: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

  - **spec.metricRelabelings.modulus**
    integer

    Modulus to take of the hash of the source label values. Only applicable when the action is `HashMod`.

  - **spec.metricRelabelings.regex**
    string

    Regular expression against which the extracted value is matched.

  - **spec.metricRelabelings.replacement**
    string

    Replacement value against which a Replace action is performed if the regular expression matches. Regex capture groups are available.

  - **spec.metricRelabelings.separator**
    string

    Separator is the string between concatenated SourceLabels.

  - **spec.metricRelabelings.sourceLabels**
    array of strings

    The source labels select values from existing labels. Their content is concatenated using the configured Separator and matched against the configured regular expression.

    - **Element of the array**
      string

      LabelName is a valid Prometheus label name which may only contain ASCII letters, numbers, as well as underscores.

Pattern: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.metricRelabelings.targetLabel**
  string

  Label to which the resulting string is written in a replacement. It is mandatory for `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` and `DropEqual` actions. Regex capture groups are available.

- **spec.metricsPath**
  string

  MetricsPath HTTP path to scrape for metrics. If empty, Prometheus uses the default value (e.g. /metrics).

- **spec.params**
  object

  Optional HTTP URL parameters

- **spec.relabelings**
  array of objects

  RelabelConfigs defines how to rewrite the target's labels before scraping. Prometheus Operator automatically adds relabelings for a few standard Kubernetes fields. The original scrape job's name is available via the `__tmp_prometheus_job_name` label. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

  RelabelConfig allows dynamic rewriting of the label set for targets, alerts, scraped samples and remote write samples. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

  - **spec.relabelings.action**
    string

    Action to perform based on the regex matching. `Uppercase` and `Lowercase` actions require Prometheus >= v2.36.0. `DropEqual` and `KeepEqual` actions require Prometheus >= v2.41.0. Default: "Replace"

    Default: `replace`

    Allowed values: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

  - **spec.relabelings.modulus**
    integer

    Modulus to take of the hash of the source label values. Only applicable when the action is `HashMod`.

  - **spec.relabelings.regex**
    string

    Regular expression against which the extracted value is matched.

  - **spec.relabelings.replacement**
    string

    Replacement value against which a Replace action is performed if the regular expression matches. Regex capture groups are available.

  - **spec.relabelings.separator**
    string

    Separator is the string between concatenated SourceLabels.

  - **spec.relabelings.sourceLabels**
    array of strings

    The source labels select values from existing labels. Their content is concatenated using the configured Separator and matched against the configured regular expression.

    - **Element of the array**
      string

      LabelName is a valid Prometheus label name which may only contain ASCII letters, numbers, as well as underscores.

      Pattern: `^[a-zA-Z_][a-zA-Z0-9_]*$`

  - **spec.relabelings.targetLabel**

string

Label to which the resulting string is written in a replacement. It is mandatory for `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` and `DropEqual` actions. Regex capture groups are available.

- **spec.sampleLimit**
integer

SampleLimit defines per-scrape limit on number of scraped samples that will be accepted.

- **spec.scheme**
string

Configures the protocol scheme used for requests. If empty, Prometheus uses HTTP by default.

Allowed values: `HTTP`, `HTTPS`

- **spec.scrapeInterval**
string

ScrapeInterval is the interval between consecutive scrapes.

Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

- **spec.scrapeTimeout**
string

ScrapeTimeout is the number of seconds to wait until a scrape request times out.

Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

- **spec.staticConfigs**
array of objects

StaticConfigs defines a list of static targets with a common label set.

StaticConfig defines a Prometheus static configuration. See https://prometheus.io/docs/prometheus/latest/configuration/configuration/#scrape_config

  - **spec.staticConfigs.labels**
  object

  Labels assigned to all metrics scraped from the targets.

  - **spec.staticConfigs.targets**
  array of strings

  List of targets for this static configuration.

    - **Element of the array**
    string

    Target represents a target for Prometheus to scrape

- **spec.targetLimit**
integer

TargetLimit defines a limit on the number of scraped targets that will be accepted.

- **spec.tlsConfig**
object

TLS configuration to use on every scrape request

  - **spec.tlsConfig.ca**
  object

  Certificate authority used when verifying server certificates.

    - **spec.tlsConfig.ca.configMap**
    object

    ConfigMap containing data to use for the targets.

      - **spec.tlsConfig.ca.configMap.key**
      string

      Required value

      The key to select.

- **spec.tlsConfig.ca.configMap.name**
  string

  Name of the referent. More info:
  https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.tlsConfig.ca.configMap.optional**
  boolean

  Specify whether the ConfigMap or its key must be defined

- **spec.tlsConfig.ca.secret**
  object

  Secret containing data to use for the targets.

  - **spec.tlsConfig.ca.secret.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.tlsConfig.ca.secret.name**
    string

    Name of the referent. More info:
    https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.tlsConfig.ca.secret.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.tlsConfig.cert**
  object

  Client certificate to present when doing client-authentication.

  - **spec.tlsConfig.cert.configMap**
    object

    ConfigMap containing data to use for the targets.

    - **spec.tlsConfig.cert.configMap.key**
      string

      Required value

      The key to select.

    - **spec.tlsConfig.cert.configMap.name**
      string

      Name of the referent. More info:
      https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.tlsConfig.cert.configMap.optional**
      boolean

      Specify whether the ConfigMap or its key must be defined

  - **spec.tlsConfig.cert.secret**
    object

    Secret containing data to use for the targets.

    - **spec.tlsConfig.cert.secret.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.tlsConfig.cert.secret.name**
      string

      Name of the referent. More info:
      https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.tlsConfig.cert.secret.optional**
      boolean

Specify whether the Secret or its key must be defined

- **spec.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

  - **spec.tlsConfig.keySecret.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.tlsConfig.keySecret.name**
    string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.tlsConfig.keySecret.optional**
    boolean

    Specify whether the Secret or its key must be defined

- **spec.tlsConfig.serverName**
  string

  Used to verify the hostname for the targets.

ServiceMonitor

Short names: smon

Scope: Namespaced
Version: v1

ServiceMonitor defines monitoring for a set of services.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

- **metadata**
  object
- **spec**
  object

  Required value

  Specification of desired Service selection for target discovery by Prometheus.

  - **spec.attachMetadata**
    object

    Attaches node metadata to discovered targets. Requires Prometheus v2.37.0 and above.

    - **spec.attachMetadata.node**
      boolean

      When set to true, Prometheus must have permissions to get Nodes.

  - **spec.endpoints**
    array of objects

    Required value

A list of endpoints allowed as part of this ServiceMonitor.

Endpoint defines a scrapeable endpoint serving Prometheus metrics.

- **spec.endpoints.authorization**
  object

  Authorization section for this endpoint

  - **spec.endpoints.authorization.credentials**
    object

    Selects a key of a Secret in the namespace that contains the credentials for authentication.

    - **spec.endpoints.authorization.credentials.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.endpoints.authorization.credentials.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.endpoints.authorization.credentials.optional**
      boolean

      Specify whether the Secret or its key must be defined

  - **spec.endpoints.authorization.type**
    string

    Defines the authentication type. The value is case-insensitive. “Basic” is not a supported value. Default: “Bearer”

- **spec.endpoints.basicAuth**
  object

  BasicAuth allow an endpoint to authenticate over basic authentication More info: https://prometheus.io/docs/operating/configuration/#endpoints

  - **spec.endpoints.basicAuth.password**
    object

    The secret in the service monitor namespace that contains the password for authentication.

    - **spec.endpoints.basicAuth.password.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.endpoints.basicAuth.password.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.endpoints.basicAuth.password.optional**
      boolean

      Specify whether the Secret or its key must be defined

  - **spec.endpoints.basicAuth.username**
    object

    The secret in the service monitor namespace that contains the username for authentication.

    - **spec.endpoints.basicAuth.username.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.endpoints.basicAuth.username.name**

string

Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.endpoints.basicAuth.username.optional**
boolean

Specify whether the Secret or its key must be defined

- **spec.endpoints.bearerTokenFile**
string

File to read bearer token for scraping targets.

- **spec.endpoints.bearerTokenSecret**
object

Secret to mount to read bearer token for scraping targets. The secret needs to be in the same namespace as the service monitor and accessible by the Prometheus Operator.

- **spec.endpoints.bearerTokenSecret.key**
string

Required value

The key of the secret to select from. Must be a valid secret key.

- **spec.endpoints.bearerTokenSecret.name**
string

Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.endpoints.bearerTokenSecret.optional**
boolean

Specify whether the Secret or its key must be defined

- **spec.endpoints.enableHttp2**
boolean

Whether to enable HTTP2.

- **spec.endpoints.filterRunning**
boolean

Drop pods that are not running. (Failed, Succeeded). Enabled by default. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#pod-phase

- **spec.endpoints.followRedirects**
boolean

FollowRedirects configures whether scrape requests follow HTTP 3xx redirects.

- **spec.endpoints.honorLabels**
boolean

HonorLabels chooses the metric’s labels on collisions with target labels.

- **spec.endpoints.honorTimestamps**
boolean

HonorTimestamps controls whether Prometheus respects the timestamps present in scraped data.

- **spec.endpoints.interval**
string

Interval at which metrics should be scraped If not specified Prometheus’ global scrape interval is used.

Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

- **spec.endpoints.metricRelabelings**
array of objects

MetricRelabelConfigs to apply to samples before ingestion.

RelabelConfig allows dynamic rewriting of the label set for targets, alerts, scraped samples and remote write samples. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

- **spec.endpoints.metricRelabelings.action**
  string

  Action to perform based on the regex matching. `Uppercase` and `Lowercase` actions require Prometheus >= v2.36.0. `DropEqual` and `KeepEqual` actions require Prometheus >= v2.41.0. Default: “Replace”

  Default: `replace`

  Allowed values: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

- **spec.endpoints.metricRelabelings.modulus**
  integer

  Modulus to take of the hash of the source label values. Only applicable when the action is `HashMod`.

- **spec.endpoints.metricRelabelings.regex**
  string

  Regular expression against which the extracted value is matched.

- **spec.endpoints.metricRelabelings.replacement**
  string

  Replacement value against which a Replace action is performed if the regular expression matches. Regex capture groups are available.

- **spec.endpoints.metricRelabelings.separator**
  string

  Separator is the string between concatenated SourceLabels.

- **spec.endpoints.metricRelabelings.sourceLabels**
  array of strings

  The source labels select values from existing labels. Their content is concatenated using the configured Separator and matched against the configured regular expression.

  - **Element of the array**
    string

    LabelName is a valid Prometheus label name which may only contain ASCII letters, numbers, as well as underscores.

    Pattern: `^[a-zA-Z_][a-zA-Z0-9_]*$`

- **spec.endpoints.metricRelabelings.targetLabel**
  string

  Label to which the resulting string is written in a replacement. It is mandatory for `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` and `DropEqual` actions. Regex capture groups are available.

- **spec.endpoints.oauth2**
  object

  OAuth2 for the URL. Only valid in Prometheus versions 2.27.0 and newer.

  - **spec.endpoints.oauth2.clientId**
    object

    Required value

    The secret or configmap containing the OAuth2 client id

    - **spec.endpoints.oauth2.clientId.configMap**
      object

      ConfigMap containing data to use for the targets.

      - **spec.endpoints.oauth2.clientId.configMap.key**
        string

        Required value

        The key to select.

      - **spec.endpoints.oauth2.clientId.configMap.name**
        string

Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

- **spec.endpoints.oauth2.clientId.configMap.optional**
boolean

  Specify whether the ConfigMap or its key must be defined

- **spec.endpoints.oauth2.clientId.secret**
object

  Secret containing data to use for the targets.

  - **spec.endpoints.oauth2.clientId.secret.key**
string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.endpoints.oauth2.clientId.secret.name**
string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.endpoints.oauth2.clientId.secret.optional**
boolean

    Specify whether the Secret or its key must be defined

- **spec.endpoints.oauth2.clientSecret**
object

  Required value

  The secret containing the OAuth2 client secret

  - **spec.endpoints.oauth2.clientSecret.key**
string

    Required value

    The key of the secret to select from. Must be a valid secret key.

  - **spec.endpoints.oauth2.clientSecret.name**
string

    Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

  - **spec.endpoints.oauth2.clientSecret.optional**
boolean

    Specify whether the Secret or its key must be defined

- **spec.endpoints.oauth2.endpointParams**
object

  Parameters to append to the token URL

- **spec.endpoints.oauth2.scopes**
array of strings

  OAuth2 scopes used for the token request

- **spec.endpoints.oauth2.tokenUrl**
string

  Required value

  The URL to fetch the token from

- **spec.endpoints.params**
object

  Optional HTTP URL parameters

- **spec.endpoints.path**
string

HTTP path to scrape for metrics. If empty, Prometheus uses the default value (e.g. `/metrics`).

- **spec.endpoints.port**
  string

  Name of the service port this endpoint refers to. Mutually exclusive with targetPort.

- **spec.endpoints.proxyUrl**
  string

  ProxyURL eg http://proxyserver:2195 Directs scrapes to proxy through this endpoint.

- **spec.endpoints.relabelings**
  array of objects

  RelabelConfigs to apply to samples before scraping. Prometheus Operator automatically adds relabelings for a few standard Kubernetes fields. The original scrape job's name is available via the `__tmp_prometheus_job_name` label. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

  RelabelConfig allows dynamic rewriting of the label set for targets, alerts, scraped samples and remote write samples. More info: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

  - **spec.endpoints.relabelings.action**
    string

    Action to perform based on the regex matching. `Uppercase` and `Lowercase` actions require Prometheus >= v2.36.0. `DropEqual` and `KeepEqual` actions require Prometheus >= v2.41.0. Default: "Replace"

    Default: `replace`

    Allowed values: `replace`, `Replace`, `keep`, `Keep`, `drop`, `Drop`, `hashmod`, `HashMod`, `labelmap`, `LabelMap`, `labeldrop`, `LabelDrop`, `labelkeep`, `LabelKeep`, `lowercase`, `Lowercase`, `uppercase`, `Uppercase`, `keepequal`, `KeepEqual`, `dropequal`, `DropEqual`

  - **spec.endpoints.relabelings.modulus**
    integer

    Modulus to take of the hash of the source label values. Only applicable when the action is `HashMod`.

  - **spec.endpoints.relabelings.regex**
    string

    Regular expression against which the extracted value is matched.

  - **spec.endpoints.relabelings.replacement**
    string

    Replacement value against which a Replace action is performed if the regular expression matches. Regex capture groups are available.

  - **spec.endpoints.relabelings.separator**
    string

    Separator is the string between concatenated SourceLabels.

  - **spec.endpoints.relabelings.sourceLabels**
    array of strings

    The source labels select values from existing labels. Their content is concatenated using the configured Separator and matched against the configured regular expression.

    - **Element of the array**
      string

      LabelName is a valid Prometheus label name which may only contain ASCII letters, numbers, as well as underscores.

      Pattern: `^[a-zA-Z_][a-zA-Z0-9_]*$`

  - **spec.endpoints.relabelings.targetLabel**
    string

    Label to which the resulting string is written in a replacement. It is mandatory for `Replace`, `HashMod`, `Lowercase`, `Uppercase`, `KeepEqual` and `DropEqual` actions. Regex capture groups are available.

- **spec.endpoints.scheme**
  string

  HTTP scheme to use for scraping. `http` and `https` are the expected values unless you rewrite the `__scheme__` label via relabeling. If empty, Prometheus uses the default value `http`.

  Allowed values: `http`, `https`

- **spec.endpoints.scrapeTimeout**
  string

  Timeout after which the scrape is ended If not specified, the Prometheus global scrape timeout is used unless it is less than `Interval` in which the latter is used.

  Pattern: `^(0|(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?)$`

- **spec.endpoints.targetPort**
  integer or string

  Name or number of the target port of the Pod behind the Service, the port must be specified with container port property. Mutually exclusive with port.

- **spec.endpoints.tlsConfig**
  object

  TLS configuration to use when scraping the endpoint

  - **spec.endpoints.tlsConfig.ca**
    object

    Certificate authority used when verifying server certificates.

    - **spec.endpoints.tlsConfig.ca.configMap**
      object

      ConfigMap containing data to use for the targets.

      - **spec.endpoints.tlsConfig.ca.configMap.key**
        string

        Required value

        The key to select.

      - **spec.endpoints.tlsConfig.ca.configMap.name**
        string

        Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

      - **spec.endpoints.tlsConfig.ca.configMap.optional**
        boolean

        Specify whether the ConfigMap or its key must be defined

    - **spec.endpoints.tlsConfig.ca.secret**
      object

      Secret containing data to use for the targets.

      - **spec.endpoints.tlsConfig.ca.secret.key**
        string

        Required value

        The key of the secret to select from. Must be a valid secret key.

      - **spec.endpoints.tlsConfig.ca.secret.name**
        string

        Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

      - **spec.endpoints.tlsConfig.ca.secret.optional**
        boolean

        Specify whether the Secret or its key must be defined

  - **spec.endpoints.tlsConfig.caFile**

string

Path to the CA cert in the Prometheus container to use for the targets.

- **spec.endpoints.tlsConfig.cert**
  object

  Client certificate to present when doing client-authentication.

  - **spec.endpoints.tlsConfig.cert.configMap**
    object

    ConfigMap containing data to use for the targets.

    - **spec.endpoints.tlsConfig.cert.configMap.key**
      string

      Required value

      The key to select.

    - **spec.endpoints.tlsConfig.cert.configMap.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.endpoints.tlsConfig.cert.configMap.optional**
      boolean

      Specify whether the ConfigMap or its key must be defined

  - **spec.endpoints.tlsConfig.cert.secret**
    object

    Secret containing data to use for the targets.

    - **spec.endpoints.tlsConfig.cert.secret.key**
      string

      Required value

      The key of the secret to select from. Must be a valid secret key.

    - **spec.endpoints.tlsConfig.cert.secret.name**
      string

      Name of the referent. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

    - **spec.endpoints.tlsConfig.cert.secret.optional**
      boolean

      Specify whether the Secret or its key must be defined

- **spec.endpoints.tlsConfig.certFile**
  string

  Path to the client cert file in the Prometheus container for the targets.

- **spec.endpoints.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.endpoints.tlsConfig.keyFile**
  string

  Path to the client key file in the Prometheus container for the targets.

- **spec.endpoints.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

  - **spec.endpoints.tlsConfig.keySecret.key**
    string

    Required value

    The key of the secret to select from. Must be a valid secret key.

        - **spec.endpoints.tlsConfig.keySecret.name**
          string

          Name of the referent. More info:
          https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

        - **spec.endpoints.tlsConfig.keySecret.optional**
          boolean

          Specify whether the Secret or its key must be defined

      - **spec.endpoints.tlsConfig.serverName**
        string

        Used to verify the hostname for the targets.

- **spec.jobLabel**
  string

  JobLabel selects the label from the associated Kubernetes service which will be used as the `job` label for all metrics. For example: If in `ServiceMonitor.spec.jobLabel: foo` and in `Service.metadata.labels.foo: bar`, then the `job="bar"` label is added to all metrics. If the value of this field is empty or if the label doesn't exist for the given Service, the `job` label of the metrics defaults to the name of the Kubernetes Service.

- **spec.keepDroppedTargets**
  integer

  Per-scrape limit on the number of targets dropped by relabeling that will be kept in memory. 0 means no limit. It requires Prometheus >= v2.47.0.

- **spec.labelLimit**
  integer

  Per-scrape limit on number of labels that will be accepted for a sample. Only valid in Prometheus versions 2.27.0 and newer.

- **spec.labelNameLengthLimit**
  integer

  Per-scrape limit on length of labels name that will be accepted for a sample. Only valid in Prometheus versions 2.27.0 and newer.

- **spec.labelValueLengthLimit**
  integer

  Per-scrape limit on length of labels value that will be accepted for a sample. Only valid in Prometheus versions 2.27.0 and newer.

- **spec.namespaceSelector**
  object

  Selector to select which namespaces the Kubernetes Endpoints objects are discovered from.

    - **spec.namespaceSelector.any**
      boolean

      Boolean describing whether all namespaces are selected in contrast to a list restricting them.

    - **spec.namespaceSelector.matchNames**
      array of strings

      List of namespace names to select from.

- **spec.podTargetLabels**
  array of strings

  PodTargetLabels transfers labels on the Kubernetes `Pod` onto the created metrics.

- **spec.sampleLimit**
  integer

  SampleLimit defines per-scrape limit on number of scraped samples that will be accepted.

- **spec.selector**
  object

  Required value

  Selector to select Endpoints objects.

    - **spec.selector.matchExpressions**
      array of objects

matchExpressions is a list of label selector requirements. The requirements are ANDed.

A label selector requirement is a selector that contains values, a key, and an operator that relates the key and values.

- **spec.selector.matchExpressions.key**
  string

  Required value

  key is the label key that the selector applies to.

- **spec.selector.matchExpressions.operator**
  string

  Required value

  operator represents a key's relationship to a set of values. Valid operators are In, NotIn, Exists and DoesNotExist.

- **spec.selector.matchExpressions.values**
  array of strings

  values is an array of string values. If the operator is In or NotIn, the values array must be non-empty. If the operator is Exists or DoesNotExist, the values array must be empty. This array is replaced during a strategic merge patch.

- **spec.selector.matchLabels**
  object

  matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels map is equivalent to an element of matchExpressions, whose key field is "key", the operator is "In", and the values array contains only "value". The requirements are ANDed.

- **spec.targetLabels**
  array of strings

  TargetLabels transfers labels from the Kubernetes `Service` onto the created metrics.

- **spec.targetLimit**
  integer

  TargetLimit defines a limit on the number of scraped targets that will be accepted.

The Prometheus monitoring module

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

> **i** Web interfaces associated with the module: grafana

The module expands the monitoring stack with preset parameters for DKP and applications, which simplifies the initial configuration.

Module features:

- The package includes ready—made triggers and dashboards, and supports push and pull models for collecting metrics. The load is optimized by using caches and Dekhouse Prom++.
- It is possible to store historical data using downsampling.
- The module covers all the basic tasks of basic monitoring of the platform and applications.

Monitoring hardware resources

Prometheus tracks hardware resources and displays the corresponding utilization graphs for:

- CPU
- Memory
- Disk drive
- Network

The graphs can be grouped by:

- Pods
- Controllers
- Namespaces
- Nodes

Kubernetes monitoring

Deckhouse configures monitoring of Kubernetes “health” parameters and its components such as:

- General cluster utilization.
- Connectivity of Kubernetes nodes to each other (RTT between all nodes is measured).
- Availability and operability of control plane components:
  - `etcd`.
  - `coredns`.
  - `kube-dns`.
  - `kube-apiserver` and others.
- Time synchronization on nodes and other parameters.

Ingress monitoring

The detailed description is available here.

Advanced monitoring mode

Deckhouse also provides the advanced monitoring mode that implements additional metrics-based alerts, such as: free space and inode-related, the node usage, the availability of Pods and container images, certificates expiration and other Kubernetes cluster events.

Alerting in advanced monitoring mode

Deckhouse allows you to flexibly configure alerts for each namespace and specify different levels of severity based on the threshold. You can set thresholds in various namespaces for the following parameters:

- Empty space and inodes on a disk.
- CPU usage for a node and a container.
- Percent of `5xx` errors on `ingress-nginx`.
- Number of unavailable Pods in a `Deployment`, `StatefulSet`, `DaemonSet`.

Monitoring Management as code (IaC approach)

The capabilities of the DKP monitoring system can be expanded by using the Observability module. It is used to implement:

- managing alerts;
- differentiation of access rights to settings and monitoring data;
- centralized dashboard management.

Alerts

The Deckhouse monitoring includes event notifications. The standard edition includes a set of basic alerts covering the health of the cluster and its components. Also, you can add custom alerts.

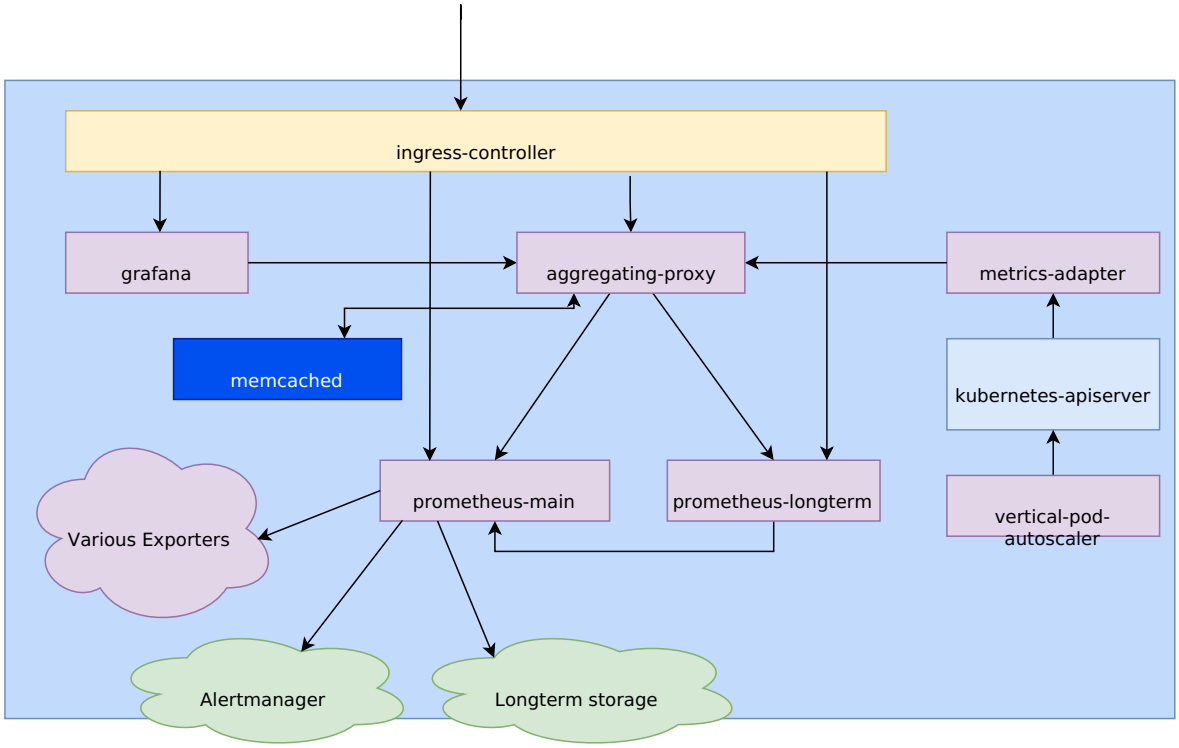
For the list of all available alerts in the Deckhouse Kubernetes Platform monitoring system, refer to the corresponding documentation page.

Sending alerts to external systems

Deckhouse supports sending alerts using `Alertmanager`:

- Via the SMTP protocol
- To PagerDuty
- To Slack
- To Telegram
- Via the Webhook mechanism
- By any other means supported in Alertmanager

Architecture



Basic monitoring components

Name	Description
prometheus-main	The primary Prometheus instance that scrapes metrics every 30 seconds (you can change this value using the `scrapeInterval` parameter). It processes all the rules, sends alerts, and serves as the main data source.
prometheus-longterm	The secondary Prometheus instance that scrapes the data of the primary Prometheus instance (`prometheus-main`) every 5 minutes (you can change this value using the `longtermScrapeInterval` parameter). It is used for long-term history storage and displaying data for large periods.
trickster	The caching proxy that reduces the load on Prometheus.
aggregating-proxy	An aggregating and caching proxy that reduces the load on Prometheus and aggregate both main and longterm in single datasource.
memcached	Distributed memory caching system.
grafana	The managed observability platform with ready-to-use dashboards for all Deckhouse modules and popular applications. Grafana instances are highly available, stateless, and configured by CRDs.
metrics-adapter	The component connecting Prometheus and Kubernetes metrics API. It enables HPA support in a Kubernetes cluster.
vertical-pod-autoscaler	An autoscaling tool to help size Pods for the optimal CPU and memory resources required by the Pods.
Various Exporters	Precooked exporters connected to Prometheus. The list includes exporters for all necessary metrics: `kube-state-metrics`, `node-exporter`, `oomkill-exporter`, `image-availability-exporter`, and many more.

Name	Description
Push/pull the metric collection model	Monitoring uses a pull model by default. Data is collected from applications at the initiative of the monitoring system. The push model is also supported: metrics can be transmitted via the Prometheus Remote Write protocol or using the Prometheus Pushgateway.

Fault Tolerance and High Availability Monitoring (HA) mode

The monitoring module provides built-in fault tolerance for all key DKP components. All monitoring services (Prometheus servers, storage systems, proxies, and other important components) are deployed in multiple copies by default. This ensures that in the event of a failure of a separate instance, the service will continue to work without loss of data and availability.

Prometheus, the main component of metric collection, runs in at least two copies (if there are enough nodes in the cluster). Both Prometheus instances use the same configuration and receive the same data. To ensure seamless operation in case of failure of one of the copies, a special component, the aggregation proxy, is used to access Prometheus. It allows you to combine metrics from both Prometheus instances and always return the most complete and up-to-date data, even if one of the copies is temporarily unavailable.

Third-party components

List of third-party software used in the `prometheus` module:

- Prometheus

  License: Apache License 2.0

  A systems and service monitoring system.

- Grafana

  License: GNU Affero General Public License v3.0

  An open-source platform for monitoring and observability.

- Trickster

  License: Apache License 2.0

  A HTTP reverse proxy/cache for http applications and a dashboard query accelerator for time series databases.

- Mimir

  License: AGPL-3.0 license

  A HTTP reverse proxy/cache for http applications and a dashboard query accelerator for time series databases.

- Promxy

  License: MIT License

  Promxy is a prometheus proxy that makes many shards of prometheus appear as a single API endpoint to the user. This significantly simplifies operations and use of prometheus at scale (when you have more than one prometheus host).

- Memcached

  License: BSD-3-Clause license

  High performance multithreaded event-based key/value cache store intended to be used in a distributed system.

Configuration

> The module lifecycle stage: General Availability
> The module has requirements for installation

The module does not require any configuration – it works right out-of-the-box.

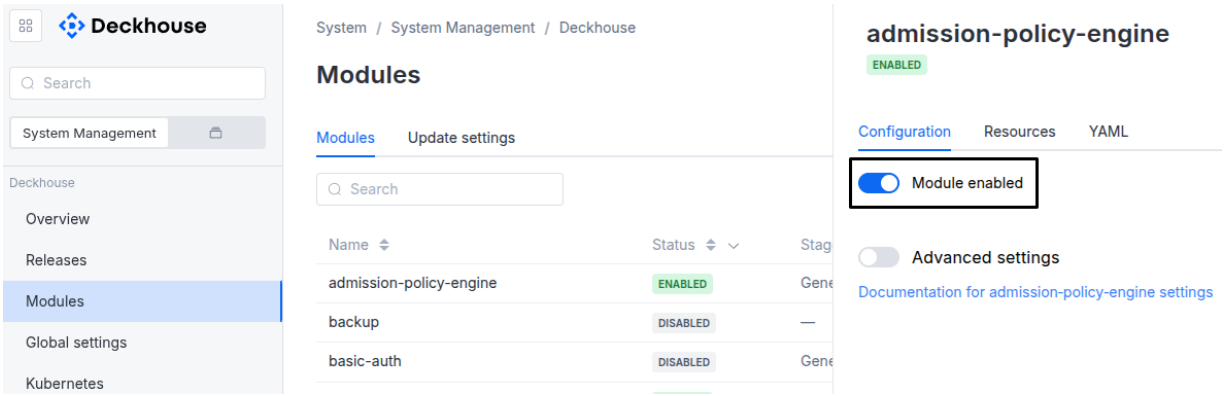
The module has 28 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `prometheus` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable prometheus
```

- **Using ModuleConfig `prometheus`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `prometheus` (create it if necessary);

  Example of a manifest to enable module `prometheus`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  enabled: true
```

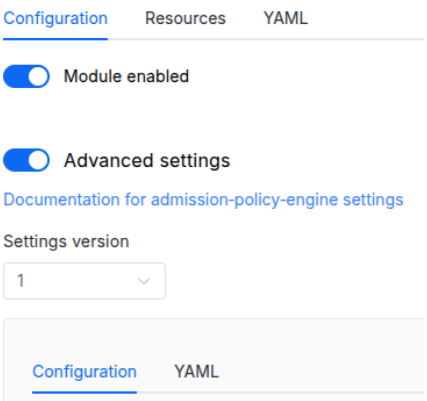
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `prometheus` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `prometheus` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `prometheus`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

Edit the existing ModuleConfig `prometheus` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc prometheus
```

Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `prometheus` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `prometheus`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Requirements

To the versions of other modules:

- `operator-prometheus`: any version.

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Remove the `.auth.password` field. If the `.auth` object becomes empty after this change, delete it.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.auth**
    object

    Options related to authentication or authorization in the application.

    - **settings.auth.allowedUserEmails**
      array of strings

      An array of emails of users that can access module’s public web interfaces.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.
    - **settings.auth.allowedUserGroups**
      array of strings

      An array of user groups that can access Grafana & Prometheus.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

      Caution! Note that you must add those groups to the appropriate field in the DexProvider config if this module is used together with the user-authn one.
    - **settings.auth.externalAuthentication**
      object

      Parameters to enable external authentication based on the Ingress NGINX external-auth mechanism that uses the NGINX auth_request module.

      > **i** External authentication is enabled automatically if the user-authn module is enabled.

      - **settings.auth.externalAuthentication.authSignInURL**
        string

        The URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response code).

        Example:

        ```
        authSignInURL: https://example.com/dex/sign_in
        ```
      - **settings.auth.externalAuthentication.authURL**
        string

        The URL of the authentication service. If the user is authenticated, the service should return an HTTP 200 response code.

        Example:

        ```
        authURL: https://example.com/dex/auth
        ```

- **settings.auth.satisfyAny**
  boolean

  Enables single authentication.

  If used together with the whitelistSourceRanges parameter, it authorizes all the users from above networks (no need to enter a username and password).

  Default: `false`

  Example:

  ```
  satisfyAny: true
  ```

- **settings.auth.whitelistSourceRanges**
  array of strings

  An array if CIDRs that are allowed to authenticate in Grafana & Prometheus.

  Example:

  ```
  whitelistSourceRanges:
  - 1.1.1.1/32
  ```

- **settings.externalLabels**
  object

  The set of external labels to add to the metrics.

  It's possible to expand the environment variables of the `config-reloader` container in external labels such as:

  - `HOSTNAME`/`POD_NAME` - contains the name of the pod (for example `prometheus-main-0`, `prometheus-main-1`, etc.).
  - `SHARD` - contains the shard number.

  Example:

  ```
  externalLabels:
    prometheus_replica: "$(POD_NAME)"
    shard: "$(SHARD)"
    hostname: "$(HOSTNAME)"
  ```

- **settings.grafana**
  object

  Grafana installation-related settings.

  - **settings.grafana.customPlugins**
    array of strings

    A list of custom Grafana plugins. Contains plugin names from the official repository.

    Here is how you can add custom plugins (in this case, clickhouse-datasource and flowcharting-panel plugins are used):

    ```
    grafana:
      customPlugins:
      - agenty-flowcharting-panel
      - vertamedia-clickhouse-datasource
    ```

    You can also install plugins from other sources by passing a link to the plugin zip archive in the format `<url to plugin zip>;<plugin name>`:

    ```
    grafana:
      customPlugins:
      - http://10.241.32.16:3000/netsage-bumpchart-panel-1.1.1.zip;netsage-bumpchart-panel
    ```

    Example:

    ```
    customPlugins:
    - agenty-flowcharting-panel
    - vertamedia-clickhouse-datasource
    ```

  - **settings.grafana.enabled**
    boolean

    Enables Grafana deploy in the cluster.

    Default: `true`

    Example:

    ```
    enabled: false
    ```

  - **settings.grafana.useDarkTheme**
    boolean

    The dark theme is enabled by default.

    Default: `false`

    Example:

    ```
    useDarkTheme: true
    ```

- **settings.highAvailability**
  boolean

  Manually enable the high availability mode.

By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

Example:

```
highAvailability: true
```

- **settings.https**
  object

  What certificate type to use with Grafana/Prometheus.

  This parameter completely overrides the `global.modules.https` settings.

  Examples:

  ```
  https:
    mode: CustomCertificate
    customCertificate:
      secretName: foobar
  ```

  ```
  https:
    mode: CertManager
    certManager:
      clusterIssuerName: letsencrypt
  ```

  - **settings.https.certManager**
    object
    - **settings.https.certManager.clusterIssuerName**
      string

      What ClusterIssuer to use for Grafana/Prometheus.

      Currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available. Also, you can define your own.

      Default: `letsencrypt`
  - **settings.https.customCertificate**
    object
    - **settings.https.customCertificate.secretName**
      string

      The name of the secret in the `d8-system` namespace to use with Grafana/Prometheus.

      This secret must have the kubernetes.io/tls format.

      Default: `false`
  - **settings.https.mode**
    string

    The HTTPS usage mode:

    - `Disabled` — Grafana/Prometheus will work over HTTP only;
    - `CertManager` — Grafana/Prometheus will use HTTPS and get a certificate from the clusterissuer defined in the `certManager.clusterIssuerName` parameter.
    - `CustomCertificate` — Grafana/Prometheus will use HTTPS using the certificate from the `d8-system` namespace.
    - `OnlyInURI` — Grafana/Prometheus will work over HTTP (thinking that there is an external HTTPS load balancer in front that terminates HTTPS traffic). All the links in the `user-authn` will be generated using the HTTPS scheme. Load balancer should provide a redirect from HTTP to HTTPS.

    Default: `Disabled`

    Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller used for Grafana/Prometheus.

  An optional parameter. By default, the `modules.ingressClass` global value is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  ingressClass: public
  ```

- **settings.longtermMaxDiskSizeGigabytes**
  Deprecated
  integer

  Deprecated and will be removed. Doesn’t affect anything.

- **settings.longtermNodeSelector**
  object

  The same as in the Pods’ `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  longtermNodeSelector:
    disktype: ssd
  ```

- **settings.longtermPodAntiAffinity**
  string

  Defines the podAntiAffinity configuration for the Prometheus longterm instance relative to the Prometheus main instance.

  - `Preferred` — allows scheduling Prometheus longterm instance alongside the Prometheus main instance if it is not possible to

place them on different nodes.
- `Required` — does not allow scheduling Prometheus longterm instance on the same node as the Prometheus main instance.

Default: `Preferred`

Allowed values: `Preferred`, `Required`

- **settings.longtermRetentionDays**
integer

  How long to keep the data in longterm Prometheus.

  Setting this parameter to `0` will result in Longterm Prometheus not running in the cluster.

  Default: `1095`

- **settings.longtermScrapeInterval**
string

  Sets the interval for making “data snapshots” of the main Prometheus by the longterm Prometheus.

  Default: `5m`

- **settings.longtermStorageClass**
string

  The name of the StorageClass to use for Longterm Prometheus.

  If omitted, the StorageClass of the existing Longterm Prometheus PVC is used. If there is no PVC yet, the StorageClass will be used according to the global storageClass parameter setting.

  The global `storageClass` parameter is only considered when the module is enabled. Changing the global `storageClass` parameter while the module is enabled will not trigger disk re-provisioning.

  Warning. Specifying a value different from the one currently used (in the existing PVC) will result in disk re-provisioning and all data will be deleted.

  Warning. When migrating Prometheus with local storage to other nodes, the pod will hang in the Pending state. In this case, it will be necessary to save the Prometheus database, delete the old PVC and restart the pod manually. Local storage refers to a StorageClass that is associated not with network storage, but with a local volume on a node (for example, StorageClass created by the local-path-provider module).

  If `false` is specified, emptyDir will be forced to be used.

  Example:

  ```
  longtermStorageClass: ceph-ssd
  ```

- **settings.longtermTolerations**
array of objects

  The same as in the Pods’ `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  longtermTolerations:
  - key: key1
    operator: Equal
    value: value1
    effect: NoSchedule
  ```

  - **settings.longtermTolerations.effect**
  string
  - **settings.longtermTolerations.key**
  string
  - **settings.longtermTolerations.operator**
  string
  - **settings.longtermTolerations.tolerationSeconds**
  integer
  - **settings.longtermTolerations.value**
  string
- **settings.mainMaxDiskSizeGigabytes**
Deprecated
integer

  Deprecated and will be removed. Doesn’t affect anything.

- **settings.nodeSelector**
object

  The same as in the Pods’ `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  nodeSelector:
    disktype: ssd
  ```

- **settings.retentionDays**
integer

  How long to keep the data.

  Default: `15`

- **settings.scrapeInterval**
string

  Sets the interval for scraping metrics from targets.

  Evaluation Interval is always equal to scrapeInterval.

Default: `30s`

Pattern: `^([\d]*y)?([\d]*w)?([\d]*d)?([\d]*h)?([\d]*m)?([\d]*s)?$`

- **settings.storageClass**
  string

  The name of the StorageClass to use for Prometheus.

  If omitted, the StorageClass of the existing Prometheus PVC is used. If there is no PVC yet, the StorageClass will be used according to the global storageClass parameter setting.

  The global `storageClass` parameter is only considered when the module is enabled. Changing the global `storageClass` parameter while the module is enabled will not trigger disk re-provisioning.

  Warning. Specifying a value different from the one currently used (in the existing PVC) will result in disk re-provisioning and all data will be deleted.

  If `false` is specified, `emptyDir` will be forced to be used.

  Examples:

  ```
  storageClass: ceph-ssd
  ```

  ```
  storageClass: 'false'
  ```

- **settings.tolerations**
  array of objects

  The same as in the Pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  tolerations:
  - key: key1
    operator: Equal
    value: value1
    effect: NoSchedule
  ```

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string
- **settings.vpa**
  object

  VPA settings for pods.

  Default: `{"updateMode":"Initial"}`

  Examples:

  ```
  vpa:
    updateMode: Initial
    longtermMaxCPU: '1'
    longtermMaxMemory: 1500Mi
    maxCPU: 1000m
    maxMemory: 1500Mi
  ```

  ```
  vpa:
    updateMode: 'Off'
  ```

  - **settings.vpa.longtermMaxCPU**
    string or number

    The maximum value that the VPA can set for the Longterm Prometheus Pods.

    The default value is chosen automatically based on the maximum number of Pods that can be created in the cluster considering the current number of nodes and their settings. For more information, see the `detect_vpa_max` hook of the module.

    Pattern: `^[0-9]+m?$`

    Example:

    ```
    longtermMaxCPU: 0.1
    ```

  - **settings.vpa.longtermMaxMemory**
    string or number

    The maximum memory requests the VPA can set for the longterm Prometheus Pods.

    The default value is chosen automatically based on the maximum number of Pods that can be created in the cluster considering the current number of nodes and their settings. For more information, see the `detect_vpa_max` hook of the module.

    Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

    Example:

    ```
    longtermMaxMemory: 4Mi
    ```

- **settings.vpa.maxCPU**
  string or number

  The maximum value that the VPA can set for the CPU requests for the main Prometheus Pods.

  The default value is chosen automatically based on the maximum number of Pods that can be created in the cluster considering the current number of nodes and their settings. For more information, see the `detect_vpa_max` hook of the module.

  Pattern: `^[0-9]+m?$`

  Example:

  ```
  maxCPU: '3'
  ```

- **settings.vpa.maxMemory**
  string or number

  The maximum memory requests the VPA can set for the main Prometheus Pods.

  The default value is chosen automatically based on the maximum number of Pods that can be created in the cluster considering the current number of nodes and their settings. For more information, see the `detect_vpa_max` hook of the module.

  Pattern: `^[0-9]+(\.[0-9]+)?(E|P|T|G|M|k|Ei|Pi|Ti|Gi|Mi|Ki)?$`

  Example:

  ```
  maxMemory: 3Mi
  ```

- **settings.vpa.updateMode**
  string

  The VPA usage mode.

  Default: `Initial`

  Allowed values: `Initial`, `Auto`, `Off`

Authentication

user-authn module provides authentication by default. Also, externalAuthentication can be configured (see below). If these options are disabled, the module will use basic auth with the auto-generated password and the user `admin`.

Use d8 k to see password:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values prometheus -o json | jq
'.internal.auth.password'
```

Delete the Secret to re-generate password:

```
d8 k -n d8-monitoring delete secret/basic-auth
```

> **Note!** The `auth.password` parameter is deprecated.

Notes

- `retentionSize` for the `main` and `longterm` Prometheus is **calculated automatically; you cannot set this value manually!**
  - The following calculation algorithm is used:
    - `pvc size * 0.85` — if the PVC exists;
    - `10 GiB` — if there is no PVC and if the StorageClass supports resizing;
    - `25 GiB` — if there is no PVC and if the StorageClass does not support resizing;
  - If the `local-storage` is used, and you have to change the `retentionSize`, then you need to manually change the size of the PV and PVC. **Caution!** Note that the value from `.status.capacity.storage` PVC is used for the calculation since it reflects the actual size of the PV in the case of manual resizing.
- `40 GiB` — size of PersistentVolumeClaim created by default.
- You can change the size of Prometheus disks in the standard Kubernetes way (if the StorageClass permits this) by editing the `.spec.resources.requests.storage` field of the PersistentVolumeClaim resource.

Custom resources

> The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability
> The module has requirements for installation

ClusterAlert

Scope: Cluster
Version: v1alpha1

Resource to display Prometheus alerts.

- **metadata**
  object
  - **metadata.name**
    string

    Alert identifier (fingerprint) as it calculated by Alertmanager.

    Pattern: `^[a-z0-9]+$`

    Maximum length: `64`

- **status**
  object
  - **status.alertStatus**
    string

    Current alert status.
  - **status.lastUpdateTime**
    string

    Timestamp of last status update for operation.
  - **status.startsAt**
    string

    Timestamp of alert creation.
- **alert**
  object

  Required value

  Alert description.
  - **alert.annotations**
    object

    Similar to the standard `metadata.annotations` field.

    Example:

    ```
    annotations:
      summary: Certificate has expired.
    ```
  - **alert.description**
    string

    Alert description.

    Optional.
  - **alert.labels**
    object

    Required value

    Similar to the standard `metadata.labels` field.

    Example:

    ```
    labels:
      alertname: NginxIngressSslExpired
      class: ingress-nginx.deckhouse.io/nginx
    ```
  - **alert.name**
    string

    Required value

    Alert name.
  - **alert.severityLevel**
    integer or string

    Severity level.

    Optional.

    Example:

    ```
    severityLevel: '4'
    ```
  - **alert.summary**
    string

    Alert summary.

    Optional.

CustomAlertmanager

Scope: Cluster
Version: v1alpha1

The resource for connecting additional Alertmanagers to Prometheus.

- **status**
  object
  - **status.deckhouse**
    object
    - **status.deckhouse.observed**
      object

      Contains last timestamp when the resource change was noted by the operator and its checksum.

      - **status.deckhouse.observed.checkSum**
        string

        The checksum of last observed resource.
      - **status.deckhouse.observed.lastTimestamp**
        string

        Last timestamp when a change of the resource was noted by the operator.

- **status.deckhouse.processed**
  object

  Contains last timestamp when the resource was applied in the cluster by the operator and its checksum.

  - **status.deckhouse.processed.checkSum**
    string

    The checksum of last processed resource.

  - **status.deckhouse.processed.lastTimestamp**
    string

    Last timestamp when a change of the resource was processed by the operator.

- **status.deckhouse.synced**
  string

  True if last observed version of the resource was successfully applied in the cluster.

- **spec**
  object

  Required value

  - **spec.external**
    object

    Parameters for connecting an external Alertmanager.

    - **spec.external.address**
      string

      URL of an external Alertmanager.

      Pattern: `^https?://[^\s/$.?#].[^\s]*$`

      Examples:

      ```
      address: http://alerts.mycompany.com
      ```

      ```
      address: https://alerts.company.com
      ```

      ```
      address: http://192.168.1.1
      ```

    - **spec.external.auth**
      object

      Authorization properties.

      - **spec.external.auth.basic**
        object

        Basic authorization properties.

        - **spec.external.auth.basic.bearerToken**
          string

          Bearer token authorization.

        - **spec.external.auth.basic.password**
          string

          User password.

        - **spec.external.auth.basic.username**
          string

          User name.

    - **spec.external.service**
      object

      Kubernetes cluster service, which leads to deployed Alertmanager.

      - **spec.external.service.name**
        string

        Required value

        Name of an Alertmanager service.

      - **spec.external.service.namespace**
        string

        Required value

        Namespace of an Alertmanager service.

      - **spec.external.service.path**
        string

        URL prefix in a service to send alerts for.

        Default: `/`

    - **spec.external.tls**
      object

      TLS settings.

      - **spec.external.tls.ca**
        string

        CA cert to use.

      - **spec.external.tls.cert**

string

Client certificate.

- **spec.external.tls.insecureSkipVerify**
boolean

Disable certificate validation.

Default: `false`

- **spec.external.tls.key**
string

Client key.

- **spec.internal**
object

Internal Alertmanager parameters.

- **spec.internal.inhibitRules**
array of objects

List of inhibition rules.

InhibitRule defines an inhibition rule that allows to mute alerts when other alerts are already firing (details...).

- **spec.internal.inhibitRules.equal**
array of strings

Labels that must have an equal value in the source and target alert for the inhibition to take effect.

- **spec.internal.inhibitRules.sourceMatch**
array of objects

Matchers for which one or more alerts have to exist for the inhibition to take effect.

Matcher defines how to match on alert’s labels.

- **spec.internal.inhibitRules.sourceMatch.name**
string

Required value

Label to match.

- **spec.internal.inhibitRules.sourceMatch.regex**
boolean

Whether to match on equality (false) or regular-expression (true).

- **spec.internal.inhibitRules.sourceMatch.value**
string

Label value to match.

- **spec.internal.inhibitRules.targetMatch**
array of objects

Matchers that have to be fulfilled in the alerts to be muted.

Matcher defines how to match on alert’s labels.

- **spec.internal.inhibitRules.targetMatch.name**
string

Required value

Label to match.

- **spec.internal.inhibitRules.targetMatch.regex**
boolean

Whether to match on equality (false) or regular-expression (true).

- **spec.internal.inhibitRules.targetMatch.value**
string

Label value to match.

- **spec.internal.receivers**
array of objects

List of receivers.

Receiver defines one or more notification integrations.

- **spec.internal.receivers.emailConfigs**
array of objects

List of Email configurations.

EmailConfig configures notifications via Email.

- **spec.internal.receivers.emailConfigs.authIdentity**
string

The identity to use for authentication.

- **spec.internal.receivers.emailConfigs.authPassword**
object

The secret’s key that contains the password to use for authentication.

The secret needs to be in the `d8-monitoring` namespace.

- **spec.internal.receivers.emailConfigs.authPassword.key**
string

Required value

The key of the secret to select from.

- **spec.internal.receivers.emailConfigs.authPassword.name**
  string

  Name of the secret.

- **spec.internal.receivers.emailConfigs.authPassword.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.emailConfigs.authSecret**
  object

  The secret's key that contains the CRAM-MD5 secret.

  The secret needs to be in the `d8-monitoring` namespace.

  - **spec.internal.receivers.emailConfigs.authSecret.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.emailConfigs.authSecret.name**
    string

    Name of the secret.

  - **spec.internal.receivers.emailConfigs.authSecret.optional**
    boolean

    Specify whether the Secret or its key must exist.

- **spec.internal.receivers.emailConfigs.authUsername**
  string

  The username to use for authentication.

- **spec.internal.receivers.emailConfigs.from**
  string

  The sender address.

- **spec.internal.receivers.emailConfigs.headers**
  array of objects

  Further headers email header key/value pairs. Overrides any headers previously set by the notification implementation.

  KeyValue defines a (key, value) tuple.

  - **spec.internal.receivers.emailConfigs.headers.key**
    string

    Required value

    Key of the tuple.

  - **spec.internal.receivers.emailConfigs.headers.value**
    string

    Required value

    Value of the tuple.

- **spec.internal.receivers.emailConfigs.hello**
  string

  The hostname to identify to the SMTP server.

- **spec.internal.receivers.emailConfigs.html**
  string

  The HTML body of the email notification.

- **spec.internal.receivers.emailConfigs.requireTLS**
  boolean

  The SMTP TLS requirement.

- **spec.internal.receivers.emailConfigs.sendResolved**
  boolean

  Whether or not to notify about resolved alerts.

- **spec.internal.receivers.emailConfigs.smarthost**
  string

  The SMTP host through which emails are sent.

- **spec.internal.receivers.emailConfigs.text**
  string

  The text body of the email notification.

- **spec.internal.receivers.emailConfigs.tlsConfig**
  object

  TLS configuration.

  - **spec.internal.receivers.emailConfigs.tlsConfig.ca**
    object

    Struct containing the CA cert to use for the targets.

- **spec.internal.receivers.emailConfigs.tlsConfig.ca.configMap**
  object

  ConfigMap containing data to use for the targets.

  - **spec.internal.receivers.emailConfigs.tlsConfig.ca.configMap.key**
    string

    Required value

    The key to select.

  - **spec.internal.receivers.emailConfigs.tlsConfig.ca.configMap.name**
    string

    Name of the ConfigMap.

  - **spec.internal.receivers.emailConfigs.tlsConfig.ca.configMap.optional**
    boolean

    Specify whether the ConfigMap or its key must exist.

- **spec.internal.receivers.emailConfigs.tlsConfig.ca.secret**
  object

  Secret containing data to use for the targets.

  - **spec.internal.receivers.emailConfigs.tlsConfig.ca.secret.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.emailConfigs.tlsConfig.ca.secret.name**
    string

    Name of the secret.

  - **spec.internal.receivers.emailConfigs.tlsConfig.ca.secret.optional**
    boolean

    Specify whether the Secret or its key must exist.

- **spec.internal.receivers.emailConfigs.tlsConfig.cert**
  object

  Struct containing the client cert file for the targets.

  - **spec.internal.receivers.emailConfigs.tlsConfig.cert.configMap**
    object

    ConfigMap containing data to use for the targets.

    - **spec.internal.receivers.emailConfigs.tlsConfig.cert.configMap.key**
      string

      Required value

      The key to select.

    - **spec.internal.receivers.emailConfigs.tlsConfig.cert.configMap.name**
      string

      Name of the ConfigMap.

    - **spec.internal.receivers.emailConfigs.tlsConfig.cert.configMap.optional**
      boolean

      Specify whether the ConfigMap or its key must exist.

  - **spec.internal.receivers.emailConfigs.tlsConfig.cert.secret**
    object

    Secret containing data to use for the targets.

    - **spec.internal.receivers.emailConfigs.tlsConfig.cert.secret.key**
      string

      Required value

      The key of the secret to select from.

    - **spec.internal.receivers.emailConfigs.tlsConfig.cert.secret.name**
      string

      Name of the secret.

    - **spec.internal.receivers.emailConfigs.tlsConfig.cert.secret.optional**
      boolean

      Specify whether the Secret or its key must exist.

- **spec.internal.receivers.emailConfigs.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.internal.receivers.emailConfigs.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

  - **spec.internal.receivers.emailConfigs.tlsConfig.keySecret.key**
    string

    Required value

    The key of the secret to select from.

        - **spec.internal.receivers.emailConfigs.tlsConfig.keySecret.name**
          string

          Name of the secret.

        - **spec.internal.receivers.emailConfigs.tlsConfig.keySecret.optional**
          boolean

          Specify whether the Secret or its key must exist.

      - **spec.internal.receivers.emailConfigs.tlsConfig.serverName**
        string

        Used to verify the hostname for the targets.

    - **spec.internal.receivers.emailConfigs.to**
      string

      The email address to send notifications to.

- **spec.internal.receivers.name**
  string

  Required value

  Name of the receiver. Must be unique across all items from the list.

- **spec.internal.receivers.opsgenieConfigs**
  array of objects

  List of OpsGenie configurations.

  OpsGenieConfig configures notifications via OpsGenie. Details...

  - **spec.internal.receivers.opsgenieConfigs.apiKey**
    object

    The secret's key that contains the OpsGenie API key.

    The secret needs to be in the `d8-monitoring` namespace.

    - **spec.internal.receivers.opsgenieConfigs.apiKey.key**
      string

      Required value

      The key of the secret to select from.

    - **spec.internal.receivers.opsgenieConfigs.apiKey.name**
      string

      Name of the secret.

    - **spec.internal.receivers.opsgenieConfigs.apiKey.optional**
      boolean

      Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.opsgenieConfigs.apiURL**
    string

    The URL to send OpsGenie API requests to.

  - **spec.internal.receivers.opsgenieConfigs.description**
    string

    Description of the incident.

  - **spec.internal.receivers.opsgenieConfigs.details**
    array of objects

    A set of arbitrary key/value pairs that provide further detail about the incident.

    - **spec.internal.receivers.opsgenieConfigs.details.key**
      string

      Required value

      Key of the tuple.

    - **spec.internal.receivers.opsgenieConfigs.details.value**
      string

      Required value

      Value of the tuple.

  - **spec.internal.receivers.opsgenieConfigs.httpConfig**
    object

    HTTP client configuration.

    - **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth**
      object

      BasicAuth for the client.

      - **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.password**
        object

        The secret in the `d8-monitoring` namespace that contains the password for authentication.

        - **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.password.key**
          string

          Required value

          The key of the secret to select from.

        - **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.password.name**

string

Name of the secret.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.password.optional**
boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.username**
object

The secret in the `d8-monitoring` namespace that contains the username for authentication.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.username.key**
string

Required value

The key of the secret to select from.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.username.name**
string

Name of the secret.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.basicAuth.username.optional**
boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.bearerTokenSecret**
object

The secret's key that contains the bearer token to be used by the client for authentication.

The secret needs to be in the `d8-monitoring` namespace.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.bearerTokenSecret.key**
string

Required value

The key of the secret to select from.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.bearerTokenSecret.name**
string

Name of the secret.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.bearerTokenSecret.optional**
boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.proxyURL**
string

Optional proxy URL.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig**
object

TLS configuration.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca**
object

Struct containing the CA cert to use for the targets.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.configMap**
object

ConfigMap containing data to use for the targets.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.configMap.key**
string

Required value

The key to select.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.configMap.name**
string

Name of the ConfigMap.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.configMap.optional**
boolean

Specify whether the ConfigMap or its key must exist.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.secret**
object

Secret containing data to use for the targets.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.secret.key**
string

Required value

The key of the secret to select from.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.secret.name**
string

Name of the secret.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.ca.secret.optional**

boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert**
  object

  Struct containing the client cert file for the targets.

  - **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.configMap**
    object

    ConfigMap containing data to use for the targets.

    - **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.configMap.key**
      string

      Required value

      The key to select.

    - **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.configMap.name**
      string

      Name of the ConfigMap.

    - **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.configMap.optional**
      boolean

      Specify whether the ConfigMap or its key must exist.

  - **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.secret**
    object

    Secret containing data to use for the targets.

    - **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.secret.key**
      string

      Required value

      The key of the secret to select from.

    - **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.secret.name**
      string

      Name of the secret.

    - **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.cert.secret.optional**
      boolean

      Specify whether the Secret or its key must exist.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

  - **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.keySecret.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.keySecret.name**
    string

    Name of the secret.

  - **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.keySecret.optional**
    boolean

    Specify whether the Secret or its key must exist.

- **spec.internal.receivers.opsgenieConfigs.httpConfig.tlsConfig.serverName**
  string

  Used to verify the hostname for the targets.

- **spec.internal.receivers.opsgenieConfigs.message**
  string

  Alert text limited to 130 characters.

- **spec.internal.receivers.opsgenieConfigs.note**
  string

  Additional alert note.

- **spec.internal.receivers.opsgenieConfigs.priority**
  string

  Priority level of alert. Possible values are P1, P2, P3, P4, and P5.

- **spec.internal.receivers.opsgenieConfigs.responders**
  array of objects

  List of responders responsible for notifications.

  OpsGenieConfigResponder defines a responder to an incident. One of `id`, `name` or `username` has to be defined.

  - **spec.internal.receivers.opsgenieConfigs.responders.id**
    string

ID of the responder.

- **spec.internal.receivers.opsgenieConfigs.responders.name**
  string

  Name of the responder.

- **spec.internal.receivers.opsgenieConfigs.responders.type**
  string

  Required value

  Type of responder.

- **spec.internal.receivers.opsgenieConfigs.responders.username**
  string

  Username of the responder.

- **spec.internal.receivers.opsgenieConfigs.sendResolved**
  boolean

  Whether or not to notify about resolved alerts.

- **spec.internal.receivers.opsgenieConfigs.source**
  string

  Backlink to the sender of the notification.

- **spec.internal.receivers.opsgenieConfigs.tags**
  string

  Comma separated list of tags attached to the notifications.

- **spec.internal.receivers.pagerdutyConfigs**
  array of objects

  List of PagerDuty configurations.

  PagerDutyConfig configures notifications via PagerDuty. Details...

  - **spec.internal.receivers.pagerdutyConfigs.class**
    string

    The class/type of the event.

  - **spec.internal.receivers.pagerdutyConfigs.client**
    string

    Client identification.

  - **spec.internal.receivers.pagerdutyConfigs.clientURL**
    string

    Backlink to the sender of notification.

  - **spec.internal.receivers.pagerdutyConfigs.component**
    string

    The part or component of the affected system that is broken.

  - **spec.internal.receivers.pagerdutyConfigs.description**
    string

    Description of the incident.

  - **spec.internal.receivers.pagerdutyConfigs.details**
    array of objects

    Arbitrary key/value pairs that provide further detail about the incident.

    - **spec.internal.receivers.pagerdutyConfigs.details.key**
      string

      Required value

      Key of the tuple.

    - **spec.internal.receivers.pagerdutyConfigs.details.value**
      string

      Required value

      Value of the tuple.

  - **spec.internal.receivers.pagerdutyConfigs.group**
    string

    A cluster or grouping of sources.

  - **spec.internal.receivers.pagerdutyConfigs.httpConfig**
    object

    HTTP client configuration.

    - **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth**
      object

      BasicAuth for the client.

      - **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.password**
        object

        The secret in the `d8-monitoring` namespace that contains the password for authentication.

        - **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.password.key**
          string

          Required value

The key of the secret to select from.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.password.name**
  string

  Name of the secret.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.password.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.username**
  object

  The secret in the `d8-monitoring` namespace that contains the username for authentication.

  - **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.username.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.username.name**
    string

    Name of the secret.

  - **spec.internal.receivers.pagerdutyConfigs.httpConfig.basicAuth.username.optional**
    boolean

    Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.bearerTokenSecret**
  object

  The secret's key that contains the bearer token to be used by the client for authentication.

  The secret needs to be in the `d8-monitoring` namespace.

  - **spec.internal.receivers.pagerdutyConfigs.httpConfig.bearerTokenSecret.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.pagerdutyConfigs.httpConfig.bearerTokenSecret.name**
    string

    Name of the secret.

  - **spec.internal.receivers.pagerdutyConfigs.httpConfig.bearerTokenSecret.optional**
    boolean

    Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.proxyURL**
  string

  Optional proxy URL.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig**
  object

  TLS configuration.

  - **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca**
    object

    Struct containing the CA cert to use for the targets.

    - **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.configMap**
      object

      ConfigMap containing data to use for the targets.

      - **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.configMap.key**
        string

        Required value

        The key to select.

      - **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.configMap.name**
        string

        Name of the ConfigMap.

      - **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.configMap.optional**
        boolean

        Specify whether the ConfigMap or its key must exist.

    - **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.secret**
      object

      Secret containing data to use for the targets.

      - **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.secret.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.secret.name**
        string

Name of the secret.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.ca.secret.optional**
boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert**
object

Struct containing the client cert file for the targets.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.configMap**
object

ConfigMap containing data to use for the targets.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.configMap.key**
string

Required value

The key to select.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.configMap.name**
string

Name of the ConfigMap.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.configMap.optional**
boolean

Specify whether the ConfigMap or its key must exist.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.secret**
object

Secret containing data to use for the targets.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.secret.key**
string

Required value

The key of the secret to select from.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.secret.name**
string

Name of the secret.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.cert.secret.optional**
boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.insecureSkipVerify**
boolean

Disable target certificate validation.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.keySecret**
object

Secret containing the client key file for the targets.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.keySecret.key**
string

Required value

The key of the secret to select from.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.keySecret.name**
string

Name of the secret.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.keySecret.optional**
boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pagerdutyConfigs.httpConfig.tlsConfig.serverName**
string

Used to verify the hostname for the targets.

- **spec.internal.receivers.pagerdutyConfigs.routingKey**
object

The secret’s key that contains the PagerDuty integration key (when using Events API v2).

Either this field or `serviceKey` needs to be defined.

The secret needs to be in the `d8-monitoring` namespace.

- **spec.internal.receivers.pagerdutyConfigs.routingKey.key**
string

Required value

The key of the secret to select from.

- **spec.internal.receivers.pagerdutyConfigs.routingKey.name**
string

Name of the secret.

- **spec.internal.receivers.pagerdutyConfigs.routingKey.optional**

boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pagerdutyConfigs.sendResolved**
boolean

Whether or not to notify about resolved alerts.

- **spec.internal.receivers.pagerdutyConfigs.serviceKey**
object

The secret's key that contains the PagerDuty service key (when using integration type "Prometheus").

Either this field or `routingKey` needs to be defined.

The secret needs to be in the `d8-monitoring` namespace.

- **spec.internal.receivers.pagerdutyConfigs.serviceKey.key**
string

Required value

The key of the secret to select from.

- **spec.internal.receivers.pagerdutyConfigs.serviceKey.name**
string

Name of the secret.

- **spec.internal.receivers.pagerdutyConfigs.serviceKey.optional**
boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pagerdutyConfigs.severity**
string

Severity of the incident.

- **spec.internal.receivers.pagerdutyConfigs.url**
string

The URL to send requests to.

- **spec.internal.receivers.pushoverConfigs**
array of objects

List of Pushover configurations.

PushoverConfig configures notifications via Pushover. Details...

- **spec.internal.receivers.pushoverConfigs.expire**
string

How long your notification will continue to be retried for, unless the user acknowledges the notification.

- **spec.internal.receivers.pushoverConfigs.html**
boolean

Whether notification message is HTML or plain text.

- **spec.internal.receivers.pushoverConfigs.httpConfig**
object

HTTP client configuration.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth**
object

BasicAuth for the client.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.password**
object

The secret in the `d8-monitoring` namespace that contains the password for authentication.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.password.key**
string

Required value

The key of the secret to select from.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.password.name**
string

Name of the secret.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.password.optional**
boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.username**
object

The secret in the `d8-monitoring` namespace that contains the username for authentication.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.username.key**
string

Required value

The key of the secret to select from.

- **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.username.name**
string

Name of the secret.

        - **spec.internal.receivers.pushoverConfigs.httpConfig.basicAuth.username.optional**
          boolean

          Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pushoverConfigs.httpConfig.bearerTokenSecret**
  object

  The secret's key that contains the bearer token to be used by the client for authentication.

  The secret needs to be in the `d8-monitoring` namespace.

    - **spec.internal.receivers.pushoverConfigs.httpConfig.bearerTokenSecret.key**
      string

      Required value

      The key of the secret to select from.

    - **spec.internal.receivers.pushoverConfigs.httpConfig.bearerTokenSecret.name**
      string

      Name of the secret.

    - **spec.internal.receivers.pushoverConfigs.httpConfig.bearerTokenSecret.optional**
      boolean

      Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pushoverConfigs.httpConfig.proxyURL**
  string

  Optional proxy URL.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig**
  object

  TLS configuration.

    - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca**
      object

      Struct containing the CA cert to use for the targets.

        - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.configMap**
          object

          ConfigMap containing data to use for the targets.

            - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.configMap.key**
              string

              Required value

              The key to select.

            - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.configMap.name**
              string

              Name of the ConfigMap.

            - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.configMap.optional**
              boolean

              Specify whether the ConfigMap or its key must exist.

        - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.secret**
          object

          Secret containing data to use for the targets.

            - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.secret.key**
              string

              Required value

              The key of the secret to select from.

            - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.secret.name**
              string

              Name of the secret.

            - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.ca.secret.optional**
              boolean

              Specify whether the Secret or its key must exist.

    - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert**
      object

      Struct containing the client cert file for the targets.

        - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.configMap**
          object

          ConfigMap containing data to use for the targets.

            - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.configMap.key**
              string

              Required value

              The key to select.

            - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.configMap.name**
              string

              Name of the ConfigMap.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.configMap.optional**
  boolean

  Specify whether the ConfigMap or its key must exist.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.secret**
  object

  Secret containing data to use for the targets.

  - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.secret.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.secret.name**
    string

    Name of the secret.

  - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.cert.secret.optional**
    boolean

    Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

  - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.keySecret.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.keySecret.name**
    string

    Name of the secret.

  - **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.keySecret.optional**
    boolean

    Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pushoverConfigs.httpConfig.tlsConfig.serverName**
  string

  Used to verify the hostname for the targets.

- **spec.internal.receivers.pushoverConfigs.message**
  string

  Notification message.

- **spec.internal.receivers.pushoverConfigs.priority**
  string

  Priority. Details…

- **spec.internal.receivers.pushoverConfigs.retry**
  string

  How often the Pushover servers will send the same notification to the user. Must be at least 30 seconds.

- **spec.internal.receivers.pushoverConfigs.sendResolved**
  boolean

  Whether or not to notify about resolved alerts.

- **spec.internal.receivers.pushoverConfigs.sound**
  string

  The name of one of the sounds supported by device clients to override the user’s default sound choice.

- **spec.internal.receivers.pushoverConfigs.title**
  string

  Notification title.

- **spec.internal.receivers.pushoverConfigs.token**
  object

  The secret’s key that contains the registered application’s API token (details…).

  The secret needs to be in the `d8-monitoring` namespace.

  - **spec.internal.receivers.pushoverConfigs.token.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.pushoverConfigs.token.name**
    string

    Name of the secret.

  - **spec.internal.receivers.pushoverConfigs.token.optional**
    boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.pushoverConfigs.url**
  string

  A supplementary URL shown alongside the message.

- **spec.internal.receivers.pushoverConfigs.urlTitle**
  string

  A title for supplementary URL, otherwise just the URL is shown.

- **spec.internal.receivers.pushoverConfigs.userKey**
  object

  The secret’s key that contains the recipient user’s user key.

  The secret needs to be in the `d8-monitoring` namespace.

  - **spec.internal.receivers.pushoverConfigs.userKey.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.pushoverConfigs.userKey.name**
    string

    Name of the secret.

  - **spec.internal.receivers.pushoverConfigs.userKey.optional**
    boolean

    Specify whether the Secret or its key must exist.

- **spec.internal.receivers.slackConfigs**
  array of objects

  List of Slack configurations.

  Configures notifications via Slack. Details...

  - **spec.internal.receivers.slackConfigs.actions**
    array of objects

    A list of Slack actions that are sent with each notification.

    Configures a single Slack action that is sent with each notification.

    More information here... and here.

    - **spec.internal.receivers.slackConfigs.actions.confirm**
      object

      Protects users from destructive actions or particularly distinguished decisions by asking them to confirm their button click one more time. Details....

      - **spec.internal.receivers.slackConfigs.actions.confirm.dismissText**
        string

        The text label for the button to cancel the action.

      - **spec.internal.receivers.slackConfigs.actions.confirm.okText**
        string

        The text label for the button to continue with an action.

      - **spec.internal.receivers.slackConfigs.actions.confirm.text**
        string

        Required value

        Describe in detail the consequences of the action and contextualize your button text choices.

        Use a maximum of 30 characters or so for best results across form factors.

      - **spec.internal.receivers.slackConfigs.actions.confirm.title**
        string

        Brief title of the pop up window.

    - **spec.internal.receivers.slackConfigs.actions.name**
      string
    - **spec.internal.receivers.slackConfigs.actions.style**
      string
    - **spec.internal.receivers.slackConfigs.actions.text**
      string

      Required value

    - **spec.internal.receivers.slackConfigs.actions.type**
      string

      Required value

    - **spec.internal.receivers.slackConfigs.actions.url**
      string
    - **spec.internal.receivers.slackConfigs.actions.value**
      string
  - **spec.internal.receivers.slackConfigs.apiURL**
    object

    The secret’s key that contains the Slack webhook URL.

    The secret needs to be in the `d8-monitoring` namespace.

    - **spec.internal.receivers.slackConfigs.apiURL.key**
      string

Required value

The key of the secret to select from.

- **spec.internal.receivers.slackConfigs.apiURL.name**
  string

  Name of the secret.

- **spec.internal.receivers.slackConfigs.apiURL.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.slackConfigs.callbackId**
  string
- **spec.internal.receivers.slackConfigs.channel**
  string

  The channel or user to send notifications to.

- **spec.internal.receivers.slackConfigs.color**
  string
- **spec.internal.receivers.slackConfigs.fallback**
  string
- **spec.internal.receivers.slackConfigs.fields**
  array of objects

  A list of Slack fields that are sent with each notification.

  Configures a single Slack field that is sent with each notification.

  Each field must contain a title, value, and optionally, a boolean value to indicate if the field is short enough to be displayed next to other fields designated as short. More information…

  - **spec.internal.receivers.slackConfigs.fields.short**
    boolean
  - **spec.internal.receivers.slackConfigs.fields.title**
    string

    Required value

  - **spec.internal.receivers.slackConfigs.fields.value**
    string

    Required value

- **spec.internal.receivers.slackConfigs.footer**
  string
- **spec.internal.receivers.slackConfigs.httpConfig**
  object

  HTTP client configuration.

  - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth**
    object

    BasicAuth for the client.

    - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.password**
      object

      The secret in the `d8-monitoring` namespace that contains the password for authentication.

      - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.password.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.password.name**
        string

        Name of the secret.

      - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.password.optional**
        boolean

        Specify whether the Secret or its key must exist.

    - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.username**
      object

      The secret in the `d8-monitoring` namespace that contains the username for authentication.

      - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.username.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.username.name**
        string

        Name of the secret.

      - **spec.internal.receivers.slackConfigs.httpConfig.basicAuth.username.optional**
        boolean

        Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.slackConfigs.httpConfig.bearerTokenSecret**
    object

    The secret's key that contains the bearer token to be used by the client for authentication.

    The secret needs to be in the `d8-monitoring` namespace.

- **spec.internal.receivers.slackConfigs.httpConfig.bearerTokenSecret.key**
  string

  Required value

  The key of the secret to select from.

- **spec.internal.receivers.slackConfigs.httpConfig.bearerTokenSecret.name**
  string

  Name of the secret.

- **spec.internal.receivers.slackConfigs.httpConfig.bearerTokenSecret.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.slackConfigs.httpConfig.proxyURL**
  string

  Optional proxy URL.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig**
  object

  TLS configuration.

  - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca**
    object

    Struct containing the CA cert to use for the targets.

    - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.configMap**
      object

      ConfigMap containing data to use for the targets.

      - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.configMap.key**
        string

        Required value

        The key to select.

      - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.configMap.name**
        string

        Name of the ConfigMap.

      - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.configMap.optional**
        boolean

        Specify whether the ConfigMap or its key must exist.

    - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.secret**
      object

      Secret containing data to use for the targets.

      - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.secret.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.secret.name**
        string

        Name of the secret.

      - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.ca.secret.optional**
        boolean

        Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.cert**
    object

    Struct containing the client cert file for the targets.

    - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.cert.configMap**
      object

      ConfigMap containing data to use for the targets.

      - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.cert.configMap.key**
        string

        Required value

        The key to select.

      - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.cert.configMap.name**
        string

        Name of the ConfigMap.

      - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.cert.configMap.optional**
        boolean

        Specify whether the ConfigMap or its key must exist.

    - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.cert.secret**
      object

      Secret containing data to use for the targets.

      - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.cert.secret.key**
        string

Required value

The key of the secret to select from.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.cert.secret.name**
  string

  Name of the secret.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.cert.secret.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

  - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.keySecret.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.keySecret.name**
    string

    Name of the secret.

  - **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.keySecret.optional**
    boolean

    Specify whether the Secret or its key must exist.

- **spec.internal.receivers.slackConfigs.httpConfig.tlsConfig.serverName**
  string

  Used to verify the hostname for the targets.

- **spec.internal.receivers.slackConfigs.iconEmoji**
  string
- **spec.internal.receivers.slackConfigs.iconURL**
  string
- **spec.internal.receivers.slackConfigs.imageURL**
  string
- **spec.internal.receivers.slackConfigs.linkNames**
  boolean
- **spec.internal.receivers.slackConfigs.mrkdwnIn**
  array of strings
- **spec.internal.receivers.slackConfigs.pretext**
  string
- **spec.internal.receivers.slackConfigs.sendResolved**
  boolean

  Whether or not to notify about resolved alerts.

- **spec.internal.receivers.slackConfigs.shortFields**
  boolean
- **spec.internal.receivers.slackConfigs.text**
  string
- **spec.internal.receivers.slackConfigs.thumbURL**
  string
- **spec.internal.receivers.slackConfigs.title**
  string
- **spec.internal.receivers.slackConfigs.titleLink**
  string
- **spec.internal.receivers.slackConfigs.username**
  string

- **spec.internal.receivers.telegramConfigs**
  array of objects

  List of Telegram configurations.

  Configures notifications via Telegram.

  Read more in the Prometheus documentation.

  - **spec.internal.receivers.telegramConfigs.apiURL**
    string

    The Telegram API URL i.e. `https://api.telegram.org`.

    If not specified, the default API URL will be used.

  - **spec.internal.receivers.telegramConfigs.botToken**
    object

    The Secret in the `d8-monitoring` namespace that contains Telegram bot token.

    - **spec.internal.receivers.telegramConfigs.botToken.key**
      string

      Required value

      The key of the secret to select from.

    - **spec.internal.receivers.telegramConfigs.botToken.name**
      string

      Name of the secret.

- **spec.internal.receivers.telegramConfigs.botToken.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.telegramConfigs.chatID**
  integer

  The Telegram chat ID.

- **spec.internal.receivers.telegramConfigs.disableNotifications**
  boolean

  Disable telegram notifications

- **spec.internal.receivers.telegramConfigs.httpConfig**
  object

  HTTP client configuration.

  - **spec.internal.receivers.telegramConfigs.httpConfig.authorization**
    object

    Authorization header configuration for the client. This is mutually exclusive with BasicAuth and is only available starting from Alertmanager v0.22+.

    - **spec.internal.receivers.telegramConfigs.httpConfig.authorization.credentials**
      object

      The Secret in the `d8-monitoring` namespace that contains the credentials of the request.

      - **spec.internal.receivers.telegramConfigs.httpConfig.authorization.credentials.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.telegramConfigs.httpConfig.authorization.credentials.name**
        string

        Name of the secret.

      - **spec.internal.receivers.telegramConfigs.httpConfig.authorization.credentials.optional**
        boolean

        Specify whether the Secret or its key must exist.

    - **spec.internal.receivers.telegramConfigs.httpConfig.authorization.type**
      string

      Set the authentication type. Defaults to Bearer, Basic will cause an error

  - **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth**
    object

    BasicAuth for the client. This is mutually exclusive with Authorization. If both are defined, BasicAuth takes precedence.

    - **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.password**
      object

      The Secret in the `d8-monitoring` namespace that contains the password for authentication.

      - **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.password.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.password.name**
        string

        Name of the secret.

      - **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.password.optional**
        boolean

        Specify whether the Secret or its key must exist.

    - **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.username**
      object

      The Secret in the `d8-monitoring` namespace that contains the username for authentication.

      - **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.username.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.username.name**
        string

        Name of the secret.

      - **spec.internal.receivers.telegramConfigs.httpConfig.basicAuth.username.optional**
        boolean

        Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.telegramConfigs.httpConfig.bearerTokenSecret**
    object

    The Secret in the `d8-monitoring` namespace that contains the bearer token to be used by the client for authentication.

    - **spec.internal.receivers.telegramConfigs.httpConfig.bearerTokenSecret.key**

string

Required value

The key of the secret to select from.

- **spec.internal.receivers.telegramConfigs.httpConfig.bearerTokenSecret.name**
string

  Name of the secret.

- **spec.internal.receivers.telegramConfigs.httpConfig.bearerTokenSecret.optional**
boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.telegramConfigs.httpConfig.followRedirects**
boolean

  FollowRedirects specifies whether the client should follow HTTP 3xx redirects.

- **spec.internal.receivers.telegramConfigs.httpConfig.oauth2**
object

  OAuth2 client credentials used to fetch a token for the targets.

  - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId**
object

    Required value

    The Secret or configMap containing the OAuth2 client id

    - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.configMap**
object

      The ConfigMap in the `d8-monitoring` namespace that contains data to use for the targets.

      - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.configMap.key**
string

        Required value

        The key to select.

      - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.configMap.name**
string

        Name of the ConfigMap.

      - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.configMap.optional**
boolean

        Specify whether the ConfigMap or its key must exist.

    - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.secret**
object

      The Secret in the `d8-monitoring` namespace that contains data to use for the targets.

      - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.secret.key**
string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.secret.name**
string

        Name of the secret.

      - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientId.secret.optional**
boolean

        Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientSecret**
object

    Required value

    The secret in the `d8-monitoring` namespace that contains OAuth2 client data.

    - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientSecret.key**
string

      Required value

      The key of the secret to select from.

    - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientSecret.name**
string

      Name of the secret.

    - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.clientSecret.optional**
boolean

      Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.endpointParams**
object

    Parameters to append to the token URL

  - **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.scopes**
array of strings

    OAuth2 scopes used for the token request

- **spec.internal.receivers.telegramConfigs.httpConfig.oauth2.tokenUrl**
  string

  Required value

  The URL to fetch the token from

- **spec.internal.receivers.telegramConfigs.httpConfig.proxyURL**
  string

  Optional proxy URL.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig**
  object

  TLS configuration for the client.

  - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca**
    object

    Certificate authority used when verifying server certificates.

    - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.configMap**
      object

      The ConfigMap in the `d8-monitoring` namespace that contains data to use for the targets.

      - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.configMap.key**
        string

        Required value

        The key to select.

      - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.configMap.name**
        string

        Name of the ConfigMap.

      - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.configMap.optional**
        boolean

        Specify whether the ConfigMap or its key must exist.

    - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.secret**
      object

      The Secret in the `d8-monitoring` namespace that contains data to use for the targets.

      - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.secret.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.secret.name**
        string

        Name of the secret.

      - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.ca.secret.optional**
        boolean

        Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert**
    object

    Client certificate to present when doing client-authentication.

    - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.configMap**
      object

      The ConfigMap in the `d8-monitoring` namespace that contains data to use for the targets.

      - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.configMap.key**
        string

        Required value

        The key to select.

      - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.configMap.name**
        string

        Name of the ConfigMap.

      - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.configMap.optional**
        boolean

        Specify whether the ConfigMap or its key must exist.

    - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.secret**
      object

      The Secret in the `d8-monitoring` namespace that contains data to use for the targets.

      - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.secret.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.secret.name**
        string

        Name of the secret.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.cert.secret.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.keySecret**
  object

  The Secret in the `d8-monitoring` namespace that contains the client key file for the targets.

  - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.keySecret.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.keySecret.name**
    string

    Name of the secret.

  - **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.keySecret.optional**
    boolean

    Specify whether the Secret or its key must exist.

- **spec.internal.receivers.telegramConfigs.httpConfig.tlsConfig.serverName**
  string

  Used to verify the hostname for the targets.

- **spec.internal.receivers.telegramConfigs.message**
  string

  Message template

- **spec.internal.receivers.telegramConfigs.parseMode**
  string

  Parse mode for telegram message

  Allowed values: `MarkdownV2`, `Markdown`, `HTML`

- **spec.internal.receivers.telegramConfigs.sendResolved**
  boolean

  Whether to notify about resolved alerts.

- **spec.internal.receivers.victoropsConfigs**
  array of objects

  List of VictorOps configurations.

  VictorOpsConfig configures notifications via VictorOps. Details...

  - **spec.internal.receivers.victoropsConfigs.apiKey**
    object

    The secret's key that contains the API key to use when talking to the VictorOps API.

    The secret needs to be in the `d8-monitoring` namespace.

    - **spec.internal.receivers.victoropsConfigs.apiKey.key**
      string

      Required value

      The key of the secret to select from.

    - **spec.internal.receivers.victoropsConfigs.apiKey.name**
      string

      Name of the secret.

    - **spec.internal.receivers.victoropsConfigs.apiKey.optional**
      boolean

      Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.victoropsConfigs.apiUrl**
    string

    The VictorOps API URL.

  - **spec.internal.receivers.victoropsConfigs.customFields**
    array of objects

    Additional custom fields for notification.

    KeyValue defines a (key, value) tuple.

    - **spec.internal.receivers.victoropsConfigs.customFields.key**
      string

      Required value

      Key of the tuple.

    - **spec.internal.receivers.victoropsConfigs.customFields.value**
      string

      Required value

      Value of the tuple.

- **spec.internal.receivers.victoropsConfigs.entityDisplayName**
  string

  Contains summary of the alerted problem.

- **spec.internal.receivers.victoropsConfigs.httpConfig**
  object

  The HTTP client’s configuration.

  - **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth**
    object

    BasicAuth for the client.

    - **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.password**
      object

      The secret in the `d8-monitoring` namespace that contains the password for authentication.

      - **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.password.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.password.name**
        string

        Name of the secret.

      - **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.password.optional**
        boolean

        Specify whether the Secret or its key must exist.

    - **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.username**
      object

      The secret in the `d8-monitoring` namespace that contains the username for authentication.

      - **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.username.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.username.name**
        string

        Name of the secret.

      - **spec.internal.receivers.victoropsConfigs.httpConfig.basicAuth.username.optional**
        boolean

        Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.victoropsConfigs.httpConfig.bearerTokenSecret**
    object

    The secret’s key that contains the bearer token to be used by the client for authentication.

    The secret needs to be in the `d8-monitoring` namespace.

    - **spec.internal.receivers.victoropsConfigs.httpConfig.bearerTokenSecret.key**
      string

      Required value

      The key of the secret to select from.

    - **spec.internal.receivers.victoropsConfigs.httpConfig.bearerTokenSecret.name**
      string

      Name of the secret.

    - **spec.internal.receivers.victoropsConfigs.httpConfig.bearerTokenSecret.optional**
      boolean

      Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.victoropsConfigs.httpConfig.proxyURL**
    string

    Optional proxy URL.

  - **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig**
    object

    TLS configuration.

    - **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca**
      object

      Struct containing the CA cert to use for the targets.

      - **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.configMap**
        object

        ConfigMap containing data to use for the targets.

        - **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.configMap.key**
          string

          Required value

          The key to select.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.configMap.name**
  string

  Name of the ConfigMap.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.configMap.optional**
  boolean

  Specify whether the ConfigMap or its key must exist.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.secret**
  object

  Secret containing data to use for the targets.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.secret.key**
  string

  Required value

  The key of the secret to select from.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.secret.name**
  string

  Name of the secret.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.ca.secret.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.cert**
  object

  Struct containing the client cert file for the targets.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.cert.configMap**
  object

  ConfigMap containing data to use for the targets.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.cert.configMap.key**
  string

  Required value

  The key to select.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.cert.configMap.name**
  string

  Name of the ConfigMap.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.cert.configMap.optional**
  boolean

  Specify whether the ConfigMap or its key must exist.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.cert.secret**
  object

  Secret containing data to use for the targets.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.cert.secret.key**
  string

  Required value

  The key of the secret to select from.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.cert.secret.name**
  string

  Name of the secret.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.cert.secret.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.keySecret.key**
  string

  Required value

  The key of the secret to select from.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.keySecret.name**
  string

  Name of the secret.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.keySecret.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.victoropsConfigs.httpConfig.tlsConfig.serverName**
  string

Used to verify the hostname for the targets.

- **spec.internal.receivers.victoropsConfigs.messageType**
  string

  Describes the behavior of the alert (CRITICAL, WARNING, INFO).

- **spec.internal.receivers.victoropsConfigs.monitoringTool**
  string

  The monitoring tool the state message is from.

- **spec.internal.receivers.victoropsConfigs.routingKey**
  string

  A key used to map the alert to a team.

- **spec.internal.receivers.victoropsConfigs.sendResolved**
  boolean

  Whether or not to notify about resolved alerts.

- **spec.internal.receivers.victoropsConfigs.stateMessage**
  string

  Contains long explanation of the alerted problem.

- **spec.internal.receivers.webhookConfigs**
  array of objects

  List of webhook configurations.

  WebhookConfig configures notifications via a generic receiver supporting the webhook payload. Details...

  - **spec.internal.receivers.webhookConfigs.httpConfig**
    object

    HTTP client configuration.

    - **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth**
      object

      BasicAuth for the client.

      - **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.password**
        object

        The secret in the `d8-monitoring` namespace that contains the password for authentication.

        - **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.password.key**
          string

          Required value

          The key of the secret to select from.

        - **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.password.name**
          string

          Name of the secret.

        - **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.password.optional**
          boolean

          Specify whether the Secret or its key must exist.

      - **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.username**
        object

        The secret in the `d8-monitoring` namespace that contains the username for authentication.

        - **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.username.key**
          string

          Required value

          The key of the secret to select from.

        - **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.username.name**
          string

          Name of the secret.

        - **spec.internal.receivers.webhookConfigs.httpConfig.basicAuth.username.optional**
          boolean

          Specify whether the Secret or its key must exist.

    - **spec.internal.receivers.webhookConfigs.httpConfig.bearerTokenSecret**
      object

      The secret's key that contains the bearer token to be used by the client for authentication.

      The secret needs to be in the `d8-monitoring` namespace.

      - **spec.internal.receivers.webhookConfigs.httpConfig.bearerTokenSecret.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.webhookConfigs.httpConfig.bearerTokenSecret.name**
        string

        Name of the secret.

      - **spec.internal.receivers.webhookConfigs.httpConfig.bearerTokenSecret.optional**
        boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.webhookConfigs.httpConfig.proxyURL**
  string

  Optional proxy URL.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig**
  object

  TLS configuration.

  - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca**
    object

    Struct containing the CA cert to use for the targets.

    - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.configMap**
      object

      ConfigMap containing data to use for the targets.

      - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.configMap.key**
        string

        Required value

        The key to select.

      - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.configMap.name**
        string

        Name of the ConfigMap.

      - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.configMap.optional**
        boolean

        Specify whether the ConfigMap or its key must exist.

    - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.secret**
      object

      Secret containing data to use for the targets.

      - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.secret.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.secret.name**
        string

        Name of the secret.

      - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.ca.secret.optional**
        boolean

        Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert**
    object

    Struct containing the client cert file for the targets.

    - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.configMap**
      object

      ConfigMap containing data to use for the targets.

      - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.configMap.key**
        string

        Required value

        The key to select.

      - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.configMap.name**
        string

        Name of the ConfigMap.

      - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.configMap.optional**
        boolean

        Specify whether the ConfigMap or its key must exist.

    - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.secret**
      object

      Secret containing data to use for the targets.

      - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.secret.key**
        string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.secret.name**
        string

        Name of the secret.

      - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.cert.secret.optional**
        boolean

        Specify whether the Secret or its key must exist.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

  - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.keySecret.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.keySecret.name**
    string

    Name of the secret.

  - **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.keySecret.optional**
    boolean

    Specify whether the Secret or its key must exist.

- **spec.internal.receivers.webhookConfigs.httpConfig.tlsConfig.serverName**
  string

  Used to verify the hostname for the targets.

- **spec.internal.receivers.webhookConfigs.maxAlerts**
  integer

  Maximum number of alerts to be sent per webhook message.

  When 0, all alerts are included.

  Allowed values: `0 <= X`

- **spec.internal.receivers.webhookConfigs.sendResolved**
  boolean

  Whether or not to notify about resolved alerts.

- **spec.internal.receivers.webhookConfigs.url**
  string

  The URL to send HTTP POST requests to.

  `urlSecret` takes precedence over `url`.

  One of `urlSecret` and `url` should be defined.

- **spec.internal.receivers.webhookConfigs.urlSecret**
  object

  The secret’s key that contains the webhook URL to send HTTP requests to.

  `urlSecret` takes precedence over `url`. One of `urlSecret` and `url` should be defined.

  The secret needs to be in the `d8-monitoring` namespace.

  - **spec.internal.receivers.webhookConfigs.urlSecret.key**
    string

    Required value

    The key of the secret to select from.

  - **spec.internal.receivers.webhookConfigs.urlSecret.name**
    string

    Name of the secret.

  - **spec.internal.receivers.webhookConfigs.urlSecret.optional**
    boolean

    Specify whether the Secret or its key must exist.

- **spec.internal.receivers.wechatConfigs**
  array of objects

  List of WeChat configurations.

  WeChatConfig configures notifications via WeChat. Details…

  - **spec.internal.receivers.wechatConfigs.agentID**
    string
  - **spec.internal.receivers.wechatConfigs.apiSecret**
    object

    The secret’s key that contains the WeChat API key.

    The secret needs to be in the `d8-monitoring` namespace.

    - **spec.internal.receivers.wechatConfigs.apiSecret.key**
      string

      Required value

      The key of the secret to select from.

    - **spec.internal.receivers.wechatConfigs.apiSecret.name**
      string

      Name of the secret.

    - **spec.internal.receivers.wechatConfigs.apiSecret.optional**

boolean

Specify whether the Secret or its key must exist.

- **spec.internal.receivers.wechatConfigs.apiURL**
string

  The WeChat API URL.

- **spec.internal.receivers.wechatConfigs.corpID**
string

  The corp id for authentication.

- **spec.internal.receivers.wechatConfigs.httpConfig**
object

  HTTP client configuration.

  - **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth**
  object

    BasicAuth for the client.

    - **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.password**
    object

      The secret in the `d8-monitoring` namespace that contains the password for authentication.

      - **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.password.key**
      string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.password.name**
      string

        Name of the secret.

      - **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.password.optional**
      boolean

        Specify whether the Secret or its key must exist.

    - **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.username**
    object

      The secret in the `d8-monitoring` namespace that contains the username for authentication.

      - **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.username.key**
      string

        Required value

        The key of the secret to select from.

      - **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.username.name**
      string

        Name of the secret.

      - **spec.internal.receivers.wechatConfigs.httpConfig.basicAuth.username.optional**
      boolean

        Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.wechatConfigs.httpConfig.bearerTokenSecret**
  object

    The secret's key that contains the bearer token to be used by the client for authentication.

    The secret needs to be in the `d8-monitoring` namespace.

    - **spec.internal.receivers.wechatConfigs.httpConfig.bearerTokenSecret.key**
    string

      Required value

      The key of the secret to select from.

    - **spec.internal.receivers.wechatConfigs.httpConfig.bearerTokenSecret.name**
    string

      Name of the secret.

    - **spec.internal.receivers.wechatConfigs.httpConfig.bearerTokenSecret.optional**
    boolean

      Specify whether the Secret or its key must exist.

  - **spec.internal.receivers.wechatConfigs.httpConfig.proxyURL**
  string

    Optional proxy URL.

  - **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig**
  object

    TLS configuration.

    - **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca**
    object

      Struct containing the CA cert to use for the targets.

      - **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.configMap**
      object

ConfigMap containing data to use for the targets.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.configMap.key**
  string

  Required value

  The key to select.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.configMap.name**
  string

  Name of the ConfigMap.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.configMap.optional**
  boolean

  Specify whether the ConfigMap or its key must exist.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.secret**
  object

  Secret containing data to use for the targets.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.secret.key**
  string

  Required value

  The key of the secret to select from.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.secret.name**
  string

  Name of the secret.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.ca.secret.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert**
  object

  Struct containing the client cert file for the targets.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.configMap**
  object

  ConfigMap containing data to use for the targets.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.configMap.key**
  string

  Required value

  The key to select.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.configMap.name**
  string

  Name of the ConfigMap.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.configMap.optional**
  boolean

  Specify whether the ConfigMap or its key must exist.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.secret**
  object

  Secret containing data to use for the targets.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.secret.key**
  string

  Required value

  The key of the secret to select from.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.secret.name**
  string

  Name of the secret.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.cert.secret.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.insecureSkipVerify**
  boolean

  Disable target certificate validation.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.keySecret**
  object

  Secret containing the client key file for the targets.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.keySecret.key**
  string

  Required value

  The key of the secret to select from.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.keySecret.name**
  string

Name of the secret.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.keySecret.optional**
  boolean

  Specify whether the Secret or its key must exist.

- **spec.internal.receivers.wechatConfigs.httpConfig.tlsConfig.serverName**
  string

  Used to verify the hostname for the targets.

- **spec.internal.receivers.wechatConfigs.message**
  string

  API request data as defined by the WeChat API.

- **spec.internal.receivers.wechatConfigs.messageType**
  string
- **spec.internal.receivers.wechatConfigs.sendResolved**
  boolean

  Whether or not to notify about resolved alerts.

- **spec.internal.receivers.wechatConfigs.toParty**
  string
- **spec.internal.receivers.wechatConfigs.toTag**
  string
- **spec.internal.receivers.wechatConfigs.toUser**
  string

- **spec.internal.route**
  object

  Required value

  The Alertmanager route definition for alerts matching the resource’s namespace. If present, it will be added to the generated Alertmanager configuration as a first-level route.

  - **spec.internal.route.continue**
    boolean

    Whether an alert should continue matching subsequent sibling nodes.

    It will always be overridden to true for the first-level route by the Prometheus operator.

  - **spec.internal.route.groupBy**
    array of strings

    List of labels to group by.

  - **spec.internal.route.groupInterval**
    string

    How long to wait before sending an updated notification.

    Must match the regular expression `[0-9]+(ms|s|m|h)` (milliseconds, seconds, minutes, hours).

  - **spec.internal.route.groupWait**
    string

    How long to wait before sending the initial notification.

    Must match the regular expression `[0-9]+(ms|s|m|h)` (milliseconds, seconds, minutes, hours).

  - **spec.internal.route.matchers**
    array of objects

    List of matchers that the alert’s labels should match.

    For the first level route, the operator removes any existing equality and regexp matcher on the `namespace` label and adds a `namespace: <object namespace>` matcher.

    Matcher defines how to match on alert’s labels.

    - **spec.internal.route.matchers.name**
      string

      Required value

      Label to match.

    - **spec.internal.route.matchers.regex**
      boolean

      Whether to match on equality (false) or regular-expression (true).

    - **spec.internal.route.matchers.value**
      string

      Label value to match.

  - **spec.internal.route.receiver**
    string

    Name of the receiver for this route.

    If not empty, it should be listed in the `receivers` field.

  - **spec.internal.route.repeatInterval**
    string

    How long to wait before repeating the last notification.

    Must match the regular expression `[0-9]+(ms|s|m|h)` (milliseconds, seconds, minutes, hours).

  - **spec.internal.route.routes**
    array

    Child routes.

    - **spec.type**
      string

      Type of the Alertmanager.

      Allowed values: `External`, `Internal`

CustomPrometheusRules

Scope: Cluster

- v1
- v1alpha1

Resource to store `PrometheusRule`. Details....

- **spec**
  object

  Required value

    - **spec.groups**
      array of objects

      Required value

      Alert group description.

      The structure of the groups is similar to that of prometheus-operator.

      RuleGroup is a list of sequentially evaluated recording and alerting rules.

        - **spec.groups.interval**
          string
        - **spec.groups.name**
          string

          Required value

        - **spec.groups.rules**
          array of objects

          Required value

            - **spec.groups.rules.alert**
              string
            - **spec.groups.rules.annotations**
              object
            - **spec.groups.rules.expr**
              integer or string

              Required value

            - **spec.groups.rules.for**
              string
            - **spec.groups.rules.labels**
              object
            - **spec.groups.rules.record**
              string

Deprecated resource. Support for the resource might be removed in a later release.

Resource to store `PrometheusRule`. Details....

- **spec**
  object

  Required value

    - **spec.groups**
      array of objects

      Required value

      Alert group description.

      The structure of the groups is similar to that of prometheus-operator.

      RuleGroup is a list of sequentially evaluated recording and alerting rules.

        - **spec.groups.interval**
          string
        - **spec.groups.name**
          string

          Required value

        - **spec.groups.rules**
          array of objects

          Required value

            - **spec.groups.rules.alert**
              string
            - **spec.groups.rules.annotations**
              object
            - **spec.groups.rules.expr**
              integer or string

              Required value

            - **spec.groups.rules.for**
              string
            - **spec.groups.rules.labels**
              object
            - **spec.groups.rules.record**
              string

GrafanaAdditionalDatasource

Scope: Cluster

- v1
- v1alpha1

The resource for connecting additional datasources to Grafana.

The resource parameters are described in details in the Grafana documentation.

- **spec**
  object

  Required value

  - **spec.access**
    string

    Required value

    Access mode.

    Allowed values: `Proxy`, `Direct`

  - **spec.basicAuth**
    boolean

    Enable/disable basic auth.

  - **spec.basicAuthUser**
    string

    Basic auth username.

  - **spec.database**
    string

    Database name (if used).

  - **spec.jsonData**
    object

    Fields to be converted to JSON and stored in jsonData.

  - **spec.secureJsonData**
    object

    JSON-data object to be saved encrypted.

  - **spec.type**
    string

    Required value

    Datasource type.

  - **spec.url**
    string

    Datasource URL.

    Example:

    ```
    url: http://localhost:8080
    ```

  - **spec.user**
    string

    Database username.

  - **spec.withCredentials**
    boolean

    Enable/disable with credentials headers (details...).

Deprecated resource. Support for the resource might be removed in a later release.

The resource for connecting additional datasources to Grafana.

The resource parameters are described in details in the Grafana documentation.

- **spec**
  object

  Required value

  - **spec.access**
    string

    Required value

    Access mode.

    Allowed values: `proxy`, `direct`

  - **spec.basicAuth**
    boolean

    Enable/disable basic auth.

  - **spec.basicAuthUser**
    string

    Basic auth username.

  - **spec.database**
    string

Database name (if used).

- **spec.jsonData**
  object

  Fields to be converted to JSON and stored in jsonData.

- **spec.secureJsonData**
  object

  JSON-data object to be saved encrypted.

- **spec.type**
  string

  Required value

  Datasource type.

- **spec.url**
  string

  Datasource URL.

  Example:

  ```
  url: http://localhost:8080
  ```

- **spec.user**
  string

  Database username.

- **spec.withCredentials**
  boolean

  Enable/disable with credentials headers (details...).

GrafanaAlertsChannel

Scope: Cluster
Version: v1alpha1

The resource for adding alert notification channels to Grafana.

- **spec**
  object

  Required value

  - **spec.alertManager**
    object

    Required value

    Alert manager settings.

    - **spec.alertManager.address**
      string

      Required value

      URL of an external Alertmanager.

      Pattern: `^https?://[^\s/$.?#].[^\s]*$`

      Examples:

      ```
      address: http://alerts.mycompany.com
      ```

      ```
      address: https://alerts.company.com
      ```

      ```
      address: http://192.168.1.1
      ```

    - **spec.alertManager.auth**
      object

      Authorization properties.

      - **spec.alertManager.auth.basic**
        object

        Required value

        Basic authorization properties.

        - **spec.alertManager.auth.basic.password**
          string

          Required value

          Password.

        - **spec.alertManager.auth.basic.username**
          string

          Required value

          User name.

  - **spec.description**
    string

    Small and optional description about this alerts channel.

Default:

- **spec.disableResolveMessage**
  boolean

  When true, this option disables the resolve message [OK] that is sent when the alerting state returns to false.

  Default: `false`

- **spec.isDefault**
  boolean

  Use this notification channel for all alerts.

  Default: `false`

- **spec.type**
  string

  Required value

  Allowed values: `PrometheusAlertManager`

GrafanaDashboardDefinition

Scope: Cluster

- v1
- v1alpha1

The resource for storing and managing Grafana dashboards.

- **spec**
  object

  Required value

  - **spec.definition**
    string

    Required value

    The dashboard’s JSON manifest.

    > **Caution!** Make sure that there is no “local” `id` (at the `.id` address) in the manifest in addition to the `uid`.

    Pattern: `^[ \t\n]*\{(.|\n)*\}[ \t\n]*$`

  - **spec.folder**
    string

    Required value

    What folder to use for a dashboard. If the folder does not exist, it will be created.

    Example:

    ```
    folder: My Folder
    ```

Deprecated resource. Support for the resource might be removed in a later release.

The resource for storing and managing Grafana dashboards.

- **spec**
  object

  Required value

  - **spec.definition**
    string

    Required value

    The dashboard’s JSON manifest.

    > **Caution!** Make sure that there is no “local” `id` (at the `.id` address) in the manifest in addition to the `uid`.

    Pattern: `^[ \t\n]*\{(.|\n)*\}[ \t\n]*$`

  - **spec.folder**
    string

    Required value

    What folder to use for a dashboard. If the folder does not exist, it will be created.

    Example:

    ```
    folder: My Folder
    ```

PrometheusRemoteWrite

Scope: Cluster

- v1
- v1alpha1

Resource for including ‘remote_write’ data from the local Prometheus into a separate longterm storage (E.g: VictoriaMetrics).

A cluster can have any number of such resources.

- **spec**
  object

  Required value

  - **spec.basicAuth**
    object

    BasicAuth for the URL.

    - **spec.basicAuth.password**
      string

      Required value

      Password for the authentication.

    - **spec.basicAuth.username**
      string

      Required value

      Username for the authentication.

  - **spec.bearerToken**
    string

    Bearer token.

  - **spec.customAuthToken**
    string

    Custom token sent as a value of the X-Auth-Token header.

  - **spec.headers**
    object

    HTTP headers to include in the request.

    - **spec.headers.X-Scope-OrgID**
      string

      Required value

      The `X-Scope-OrgID` header specifying the tenant ID.

  - **spec.tlsConfig**
    object

    TLS Config to use for remote write.

    - **spec.tlsConfig.ca**
      string

      The CA certificate used when verifying the API server certificate.

      Default:

    - **spec.tlsConfig.insecureSkipVerify**
      boolean

      Disable target certificate validation.

      Default: `false`

  - **spec.url**
    string

    Required value

    The URL of the endpoint to send samples to.

    Example:

    ```
    url: https://victoriametrics-test.domain.com/api/v1/write
    ```

  - **spec.writeRelabelConfigs**
    array of objects

    The list of remote write relabel configurations.

    For example, if you need to remove unnecessary metrics or perform data relabeling.

    - **spec.writeRelabelConfigs.action**
      string

      Action to perform based on regex matching.

      Default: `replace`

    - **spec.writeRelabelConfigs.modulus**
      integer

      Modulus to take of the hash of the source label values.

    - **spec.writeRelabelConfigs.regex**
      string

      Regular expression against which the extracted value is matched. Default is '(.*)'

      Default: `(.*)`

    - **spec.writeRelabelConfigs.replacement**
      string

      Replacement value against which a regex replace is performed if the regular expression matches.

      Regex capture groups are available.

Default: `$1`

- **spec.writeRelabelConfigs.separator**
  string

  Separator placed between concatenated source label values.

  Default: `;`

- **spec.writeRelabelConfigs.sourceLabels**
  array of strings

  The source labels select values from existing labels.

  Their content is concatenated using the configured separator and matched against the configured regular expression for the replace, keep, and drop actions.

- **spec.writeRelabelConfigs.targetLabel**
  string

  Label to which the resulting value is written in a replace action. It is mandatory for replace actions.

  Regex capture groups are available.

Deprecated resource. Support for the resource might be removed in a later release.

Resource for including ‘remote_write’ data from the local Prometheus into a separate longterm storage (E.g: VictoriaMetrics).

A cluster can have any number of such resources.

- **spec**
  object

  Required value

  - **spec.basicAuth**
    object

    BasicAuth for the URL.

    - **spec.basicAuth.password**
      string

      Required value

      Password for the authentication.

    - **spec.basicAuth.username**
      string

      Required value

      Username for the authentication.

  - **spec.bearerToken**
    string

    Bearer token.

  - **spec.customAuthToken**
    string

    Custom token sent as a value of the X-Auth-Token header.

  - **spec.headers**
    object

    HTTP headers to include in the request.

    - **spec.headers.X-Scope-OrgID**
      string

      Required value

      The `X-Scope-OrgID` header specifying the tenant ID.

  - **spec.tlsConfig**
    object

    TLS Config to use for remote write.

    - **spec.tlsConfig.ca**
      string

      The CA certificate used when verifying the API server certificate.

      Default:

    - **spec.tlsConfig.insecureSkipVerify**
      boolean

      Disable target certificate validation.

      Default: `false`

  - **spec.url**
    string

    Required value

    The URL of the endpoint to send samples to.

    Example:

    ```
    url: https://victoriametrics-test.domain.com/api/v1/write
    ```

  - **spec.writeRelabelConfigs**
    array of objects

The list of remote write relabel configurations.

For example, if you need to remove unnecessary metrics or perform data relabeling.

- **spec.writeRelabelConfigs.action**
  string

  Action to perform based on regex matching.

  Default: `replace`
- **spec.writeRelabelConfigs.modulus**
  integer

  Modulus to take of the hash of the source label values.
- **spec.writeRelabelConfigs.regex**
  string

  Regular expression against which the extracted value is matched. Default is '(.*)'

  Default: `(.*)`
- **spec.writeRelabelConfigs.replacement**
  string

  Replacement value against which a regex replace is performed if the regular expression matches.

  Regex capture groups are available.

  Default: `$1`
- **spec.writeRelabelConfigs.separator**
  string

  Separator placed between concatenated source label values.

  Default: `;`
- **spec.writeRelabelConfigs.sourceLabels**
  array of strings

  The source labels select values from existing labels.

  Their content is concatenated using the configured separator and matched against the configured regular expression for the replace, keep, and drop actions.
- **spec.writeRelabelConfigs.targetLabel**
  string

  Label to which the resulting value is written in a replace action. It is mandatory for replace actions.

  Regex capture groups are available.

Usage

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

An example of the module configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  enabled: true
  settings:
    auth:
      password: xxxxxx
    retentionDays: 7
    storageClass: rbd
    nodeSelector:
      node-role/monitoring: ""
    tolerations:
    - key: dedicated.deckhouse.io
      operator: Equal
      value: monitoring
```

Writing Prometheus data to the longterm storage

Prometheus supports remote_write'ing data from the local Prometheus to a separate longterm storage (e.g., VictoriaMetrics). In Deckhouse, this mechanism is implemented using the `PrometheusRemoteWrite` custom resource.

> **i** For VictoriaMetrics detailed information about how to send data to vmagent can be found in the VictoriaMetrics documentation.

Example of the basic PrometheusRemoteWrite

```
apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: test-remote-write
spec:
  url: https://victoriametrics-test.domain.com/api/v1/write
```

Example of the expanded PrometheusRemoteWrite

```
apiVersion: deckhouse.io/v1
kind: PrometheusRemoteWrite
metadata:
  name: test-remote-write
spec:
  url: https://victoriametrics-test.domain.com/api/v1/write
  basicAuth:
    username: username
    password: password
  writeRelabelConfigs:
  - sourceLabels: [__name__]
    action: keep
    regex: prometheus_build_.*|my_cool_app_metrics_.*
  - sourceLabels: [__name__]
    action: drop
    regex: my_cool_app_metrics_with_sensitive_data
```

Connecting Prometheus to an external Grafana instance

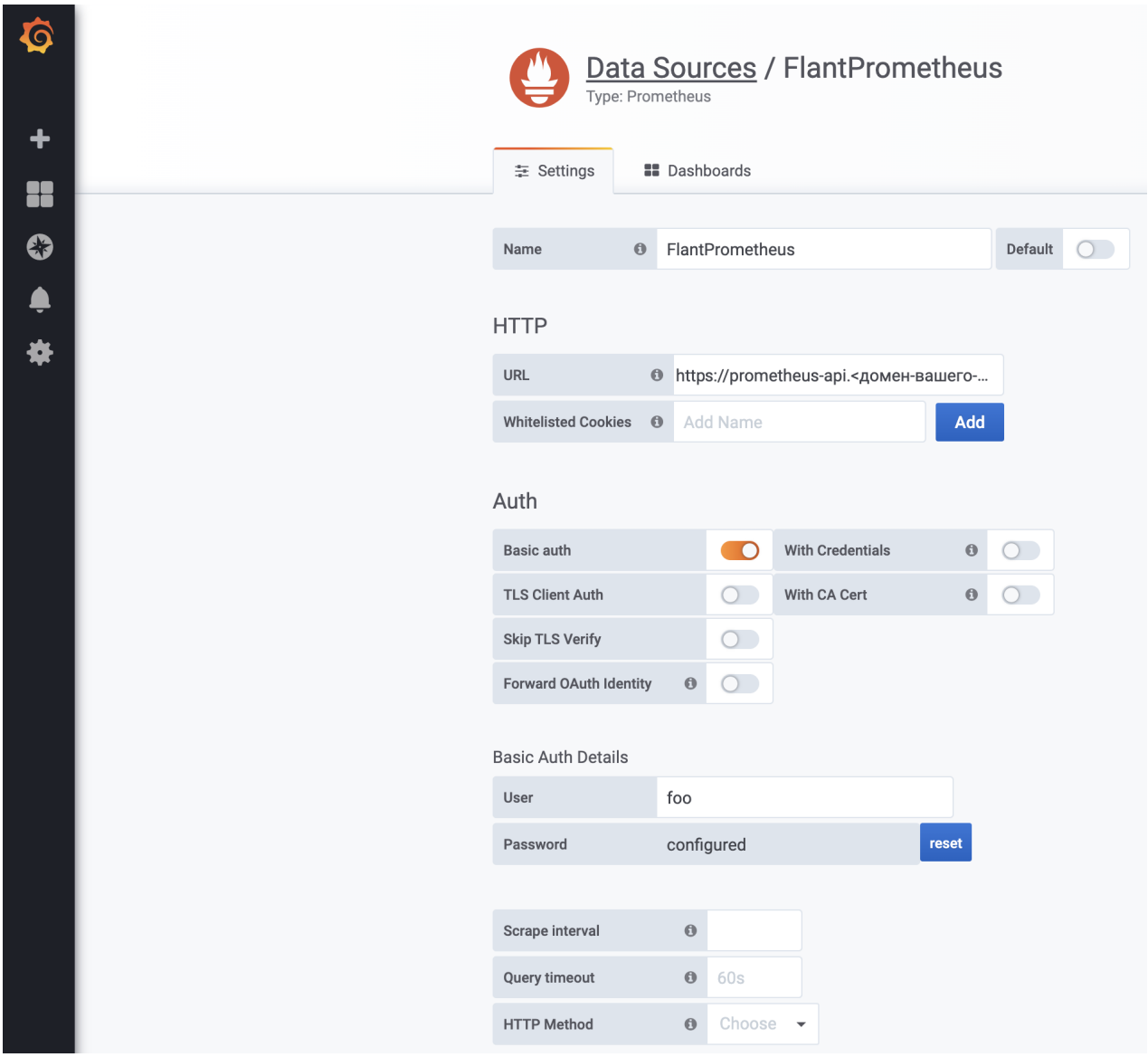
Each `ingress-nginx-controller` has certificates that can be used to connect to Prometheus. All you need is to create an additional `Ingress` resource.

> **i**
>
> For the example below, it is presumed that Secret `example-com-tls` already exist in namespace d8-monitoring.
>
> Names for Ingress `my-prometheus-api` and Secret `my-basic-auth-secret` are there for example. Change them to the most suitable names for your case.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-prometheus-api
  namespace: d8-monitoring
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
    nginx.ingress.kubernetes.io/auth-type: basic
    nginx.ingress.kubernetes.io/auth-secret: my-basic-auth-secret
    nginx.ingress.kubernetes.io/app-root: /graph
    nginx.ingress.kubernetes.io/configuration-snippet: |
      proxy_ssl_certificate /etc/nginx/ssl/client.crt;
      proxy_ssl_certificate_key /etc/nginx/ssl/client.key;
      proxy_ssl_protocols TLSv1.2;
      proxy_ssl_session_reuse on;
spec:
  ingressClassName: nginx
  rules:
  - host: prometheus-api.example.com
    http:
      paths:
      - backend:
          service:
            name: prometheus
            port:
              name: https
        path: /
        pathType: ImplementationSpecific
  tls:
  - hosts:
    - prometheus-api.example.com
    secretName: example-com-tls
---
apiVersion: v1
kind: Secret
metadata:
  name: my-basic-auth-secret
  namespace: d8-monitoring
type: Opaque
data:
  # Basic-auth string is hashed using htpasswd.
  auth: Zm9vOiRhcHIxJE9GRzNYeWJwJGNrTDBGSERBa29YWUlsSDkuY3lzVDAK  # foo:bar
```

Next, you only need to add the data source to Grafana:

> Specify `https://prometheus-api.<cluster-domain>` as the URL.



- Note that **basic authorization** is not sufficiently secure and safe. You are encouraged to implement additional safety measures, e.g., attach the `nginx.ingress.kubernetes.io/whitelist-source-range` annotation.
- It is **not recommended** connecting this way since you have to create an Ingress resource in the system namespace. Deckhouse does **not guarantee** the functionality of this connection method due to its regular updates.
- This Ingress resource can be used to access the Prometheus API not only from Grafana but for other integrations, e.g., the Prometheus federation.

Connecting an external app to Prometheus

The connection to Prometheus is protected using kube-rbac-proxy. To connect, you need to create a `ServiceAccount` with the necessary permissions.

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app:prometheus-access
rules:
- apiGroups: ["monitoring.coreos.com"]
  resources: ["prometheuses/http"]
  resourceNames: ["main", "longterm"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: app:prometheus-access
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app:prometheus-access
subjects:
- kind: ServiceAccount
  name: app
  namespace: default
```

Define the following job containing the `curl` request:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: app-curl
  namespace: default
spec:
  template:
    metadata:
      name: app-curl
    spec:
      serviceAccountName: app
      containers:
      - name: app-curl
        image: curlimages/curl:7.69.1
        command: ["sh", "-c"]
        args:
        - >-
          curl -H "Authorization: Bearer $(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" -k -f
          https://prometheus.d8-monitoring:9090/api/v1/query_range?query=up\&start=1584001500\&end=1584023100\&step=30
      restartPolicy: Never
  backoffLimit: 4
```

The `job` must complete successfully.

Sending alerts to Telegram

Alertmanager supports sending alerts to Telegram directly.

Create the Secret in the `d8-monitoring` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: telegram-bot-secret
  namespace: d8-monitoring
stringData:
  token: "562696849:AAExcuJ8H6z4pTlPuocbrXXXXXXXXXXx"
```

Deploy CustomAlertManager CR:

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: telegram
spec:
  type: Internal
  internal:
    receivers:
      - name: telegram
        telegramConfigs:
          - botToken:
              name: telegram-bot-secret
              key: token
            chatID: -30490XXXXX
    route:
      groupBy:
        - job
      groupInterval: 5m
      groupWait: 30s
      receiver: telegram
      repeatInterval: 12h
```

The fields `token` in the Secret and `chatID` in the `CustomAlertmanager` custom resource must be set on your own. Read more about Telegram API.

Example of sending alerts to Slack with a filter

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: slack
spec:
  internal:
    receivers:
    - name: devnull
    - name: slack
      slackConfigs:
      - apiURL:
          key: apiURL
          name: slack-apiurl
        channel: {{ dig .Values.werf.env .Values.slack.channel._default .Values.slack.channel }}
        fields:
        - short: true
          title: Severity
          value: '{{`{{  .CommonLabels.severity_level }}`}}'
        - short: true
          title: Status
          value: '{{`{{ .Status }}`}}'
        - title: Summary
          value: '{{`{{ range .Alerts }}`}}{{`{{ .Annotations.summary }}`}} {{`{{ end }}`}}'
        - title: Description
          value: '{{`{{ range .Alerts }}`}}{{`{{ .Annotations.description }}`}} {{`{{ end }}`}}'
        - title: Labels
          value: '{{`{{ range .Alerts }}`}} {{`{{ range .Labels.SortedPairs }}`}}{{`{{ printf "%s:
            %s\n" .Name .Value }}`}}{{`{{ end }}`}}{{`{{ end }}`}}'
        - title: Links
          value: '{{`{{ (index .Alerts 0).GeneratorURL }}`}}'
        title: '{{`{{ .CommonLabels.alertname }}`}}'
    route:
      groupBy:
      - '...'
      receiver: devnull
      routes:
        - matchers:
          - matchType: =~
            name: severity_level
            value: "^[4-9]$"
          receiver: slack
      repeatInterval: 12h
  type: Internal
```

Example of sending alerts to Opsgenie

```
- name: opsgenie
      opsgenieConfigs:
        - apiKey:
            key: data
            name: opsgenie
          description: |
            {{ range .Alerts }}{{ .Annotations.summary }} {{ end }}
            {{ range .Alerts }}{{ .Annotations.description }} {{ end }}
          message: '{{ .CommonLabels.alertname }}'
          priority: P1
          responders:
            - id: team_id
              type: team
```

Example of sending alert by Email

Create a Secret with base64 encoded password for email account as value of `password`:

```
apiVersion: v1
kind: Secret
metadata:
  name: am-mail-server-pass
  namespace: d8-monitoring
data:
  password: BASE64_ENCODED_PASSWORD_HERE
```

Change values in `CustomAlertManager` manifest as you need and apply the resource:

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: mail
spec:
  type: Internal
  internal:
    receivers:
      - name: devnull
      - name: mail
        emailConfigs:
          - to: oncall@example.com
            from: prom@example.com
            smarthost: mx.example.com:587
            authIdentity: prom@example.com
            authUsername: prom@example.com
            authPassword:
              key: password
              name: am-mail-server-pass
            # In case you have custom TLS certificates, you can put public part of your CA in a ConfigMap and deploy it
in d8-monitoring namespace
            # tlsConfig:
            #   insecureSkipVerify: true
            #   ca:
            #     configMap:
            #       key: ca.pem
            #       name: alertmanager-mail-server-ca
            sendResolved: true
            requireTLS: true
    route:
      groupBy:
        - job
      groupInterval: 5m
      groupWait: 30s
      receiver: devnull
      repeatInterval: 24h
      routes:
        - matchers:
          - matchType: =~
            name: severity_level
            value: "^[1-4]$"
          receiver: mail
```

FAQ

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

How do I collect metrics from applications running outside of the cluster?

1. Configure a Service similar to the one that collects metrics from your application (but do not set the `spec.selector` parameter).
2. Create Endpoints for this Service and explicitly specify the `IP:PORT` pairs that your applications use to expose metrics.

> **i** Port names in Endpoints must match those in the Service.

An example

Application metrics are freely available (no TLS involved) at `http://10.182.10.5:9114/metrics`.

```
apiVersion: v1
kind: Service
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    prometheus.deckhouse.io/custom-target: my-app
spec:
  ports:
  - name: http-metrics
    port: 9114
---
apiVersion: v1
kind: Endpoints
metadata:
  name: my-app
  namespace: my-namespace
subsets:
  - addresses:
    - ip: 10.182.10.5
    ports:
    - name: http-metrics
      port: 9114
```

How do I create custom Grafana dashboards?

Custom Grafana dashboards can be added to the project using the Infrastructure as a Code approach. To add your dashboard to Grafana, create the dedicated `GrafanaDashboardDefinition` Custom Resource in the cluster.

An example:

```
apiVersion: deckhouse.io/v1
kind: GrafanaDashboardDefinition
metadata:
  name: my-dashboard
spec:
  folder: My folder # The folder where the custom dashboard will be located.
  definition: |
    {
      "annotations": {
        "list": [
          {
            "builtIn": 1,
            "datasource": "-- Grafana --",
            "enable": true,
            "hide": true,
            "iconColor": "rgba(0, 211, 255, 1)",
            "limit": 100,
...
```

> **⚠** System dashboards and dashboards added using GrafanaDashboardDefinition cannot be modified via the Grafana interface.
>
> Alerts configured in the dashboard do not work with datasource templates - such a dashboard is invalid and cannot be imported. In Grafana 9.0, the legacy alerting functionality was deprecated and replaced with Grafana Alerting. Therefore, we do not recommend using legacy alerting in dashboards.

> **i** If the dashboard does not appear in Grafana after being applied, there might be an error in the dashboard's JSON file. To identify the source of the problem, use the command `d8 k logs -n d8-monitoring deployments/grafana-v10 dashboard-provisioner` to view the logs of the component responsible for applying dashboards.

How do I add alerts and/or recording rules?

There are two ways to add alerts:

- **Recommended**: Using the ClusterObservabilityMetricsRulesGroup and ObservabilityMetricsRulesGroup resources of the `observability` module.
- **Outdated**: Using the CustomPrometheusRules resource. The following is an example configuration for this resource.

Parameters:

- `groups` — is the only parameter where you need to define alert groups. The structure of the groups is similar to that of prometheus-operator.

An example:

```
apiVersion: deckhouse.io/v1
kind: CustomPrometheusRules
metadata:
  name: my-rules
spec:
  groups:
  - name: cluster-state-alert.rules
    rules:
    - alert: CephClusterErrorState
      annotations:
        description: Storage cluster is in error state for more than 10m.
        summary: Storage cluster is in error state
        plk_markup_format: markdown
      expr: |
        ceph_health_status{job="rook-ceph-mgr"} > 1
```

How do I provision additional Grafana data sources?

The `GrafanaAdditionalDatasource` allows you to provision additional Grafana data sources.

A detailed description of the resource parameters is available in the Grafana documentation.

See the datasource type in the documentation for the specific datasource.

An example:

```
apiVersion: deckhouse.io/v1
kind: GrafanaAdditionalDatasource
metadata:
  name: another-prometheus
spec:
  type: prometheus
  access: Proxy
  url: https://another-prometheus.example.com/prometheus
  basicAuth: true
  basicAuthUser: foo
  jsonData:
    timeInterval: 30s
    httpMethod: POST
  secureJsonData:
    basicAuthPassword: bar
```

How do I enable secure access to metrics?

To enable secure access to metrics, we strongly recommend using **kube-rbac-proxy**.

An example of collecting metrics securely from an application inside a cluster

To set up application metrics protection using `kube-rbac-proxy` and then collect metrics using Prometheus, follow these steps:

1. Create a new `ServiceAccount` with the following permissions:

   ```
   ---
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: rbac-proxy-test
   ---
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRoleBinding
   metadata:
     name: rbac-proxy-test
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: ClusterRole
     name: d8:rbac-proxy
   subjects:
   - kind: ServiceAccount
     name: rbac-proxy-test
     namespace: default
   ```

   > **i** The example uses the `d8:rbac-proxy` built-in Deckhouse `ClusterRole`.

2. Create a configuration for the `kube-rbac-proxy`:

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: rbac-proxy-config-test
  namespace: rbac-proxy-test
data:
  config-file.yaml: |+
    authorization:
      resourceAttributes:
        namespace: default
        apiVersion: v1
        resource: services
        subresource: proxy
        name: rbac-proxy-test
```

> **i** Get more information on authorization attributes in the Kubernetes documentation.

3. Create `Service` and `Deployment` for your application with the `kube-rbac-proxy` as a sidecar container:

```
---
apiVersion: v1
kind: Service
metadata:
  name: rbac-proxy-test
  labels:
    prometheus.deckhouse.io/custom-target: rbac-proxy-test
spec:
  ports:
  - name: https-metrics
    port: 8443
    targetPort: https-metrics
  selector:
    app: rbac-proxy-test
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rbac-proxy-test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: rbac-proxy-test
  template:
    metadata:
      labels:
        app: rbac-proxy-test
    spec:
      securityContext:
        runAsUser: 65532
      serviceAccountName: rbac-proxy-test
      containers:
      - name: kube-rbac-proxy
        image: quay.io/brancz/kube-rbac-proxy:v0.14.0
        args:
        - "--secure-listen-address=0.0.0.0:8443"
        - "--upstream=http://127.0.0.1:8081/"
        - "--config-file=/kube-rbac-proxy/config-file.yaml"
        - "--logtostderr=true"
        - "--v=10"
        ports:
        - containerPort: 8443
          name: https-metrics
        volumeMounts:
        - name: config
          mountPath: /kube-rbac-proxy
      - name: prometheus-example-app
        image: quay.io/brancz/prometheus-example-app:v0.1.0
        args:
        - "--bind=127.0.0.1:8081"
      volumes:
      - name: config
        configMap:
          name: rbac-proxy-config-test
```

4. Add the necessary resource permissions to Prometheus:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: rbac-proxy-test-client
rules:
- apiGroups: [""]
  resources: ["services/proxy"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: rbac-proxy-test-client
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: rbac-proxy-test-client
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: d8-monitoring
```

After step 4, your application's metrics should become available in Prometheus.

An example of collecting metrics securely from an application outside a cluster

Suppose there is a server exposed to the Internet on which the `node-exporter` is running. By default, the `node-exporter` listens on port `9100` and is available on all interfaces. One needs to ensure access control to the `node-exporter` so that metrics can be collected securely. Below is an example of how you can set this up.

Requirements:

- There must be network access from the cluster to the `kube-rbac-proxy` service running on the *remote server*.
- The *remote server* must have access to the Kubernetes API server.

Follow these steps:

1. Create a new `ServiceAccount` with the following permissions:

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus-external-endpoint-server-01
  namespace: d8-service-accounts
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus-external-endpoint
rules:
- apiGroups: ["authentication.k8s.io"]
  resources:
  - tokenreviews
  verbs: ["create"]
- apiGroups: ["authorization.k8s.io"]
  resources:
  - subjectaccessreviews
  verbs: ["create"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus-external-endpoint-server-01
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus-external-endpoint
subjects:
- kind: ServiceAccount
  name: prometheus-external-endpoint-server-01
  namespace: d8-service-accounts
```

2. Generate a `kubeconfig` file for the created `ServiceAccount` (refer to the example on how to generate `kubeconfig` for `ServiceAccount`).
3. Copy the `kubeconfig` file to the *remote server*. You will also have to specify the `kubeconfig` path in the `kube-rbac-proxy` settings (our example uses `${PWD}/.kube/config`).
4. Configure `node-exporter` on the *remote server* to be accessible only on the local interface (i.e., listening on `127.0.0.1:9100`).
5. Run `kube-rbac-proxy` on the *remote server*:

```
docker run --network host -d -v ${PWD}/.kube/config:/config quay.io/brancz/kube-rbac-proxy:v0.14.0 --secure-listen-
address=0.0.0.0:8443 \
  --upstream=http://127.0.0.1:9100 --kubeconfig=/config --logtostderr=true --v=10
```

6. Check that port `8443` is accessible at the remote server's external address.

7. Create `Service` and `Endpoint`, specifying the external address of the *remote server* as `<server_ip_address>`:

```
---
apiVersion: v1
kind: Service
metadata:
  name: prometheus-external-endpoint-server-01
  labels:
    prometheus.deckhouse.io/custom-target: prometheus-external-endpoint-server-01
spec:
  ports:
  - name: https-metrics
    port: 8443
---
apiVersion: v1
kind: Endpoints
metadata:
  name: prometheus-external-endpoint-server-01
subsets:
  - addresses:
    - ip: <server_ip_address>
    ports:
    - name: https-metrics
      port: 8443
```

How do I add Alertmanager?

Create a custom resource `CustomAlertmanager` with type `Internal`.

Example:

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: webhook
spec:
  type: Internal
  internal:
    route:
      groupBy: ['job']
      groupWait: 30s
      groupInterval: 5m
      repeatInterval: 12h
      receiver: 'webhook'
    receivers:
    - name: 'webhook'
      webhookConfigs:
      - url: 'http://webhookserver:8080/'
```

Refer to the description of the CustomAlertmanager custom resource for more information about the parameters.

How do I add an additional Alertmanager?

Create a custom resource `CustomAlertmanager` with the type `External`, it can point to Alertmanager through the FQDN or Kubernetes service.

FQDN Alertmanager example:

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: my-fqdn-alertmanager
spec:
  external:
    address: https://alertmanager.mycompany.com/myprefix
  type: External
```

Alertmanager with a Kubernetes service:

```
apiVersion: deckhouse.io/v1alpha1
kind: CustomAlertmanager
metadata:
  name: my-service-alertmanager
spec:
  external:
    service:
      namespace: myns
      name: my-alertmanager
      path: /myprefix/
  type: External
```

Refer to the description of the CustomAlertmanager Custom Resource for more information about the parameters.

How do I ignore unnecessary alerts in Alertmanager?

The solution comes down to configuring alert routing in the Alertmanager.

You will need to:

1. Create a parameterless receiver.
2. Route unwanted alerts to this receiver.

Below are samples for configuring `CustomAlertmanager`.

Receive all alerts with labels `service: foo|bar|baz`:

```
receivers:
 # The parameterless receiver is similar to "/dev/null".
 - name: blackhole
 # Your valid receiver.
 - name: some-other-receiver
   # ...
route:
 # Default receiver.
 receiver: blackhole
 routes:
   # Child receiver.
   - matchers:
       - matchType: =~
         name: service
         value: ^(foo|bar|baz)$
     receiver: some-other-receiver
```

Receive all alerts except for `DeadMansSwitch`:

```
receivers:
 # The parameterless receiver is similar to "/dev/null".
 - name: blackhole
 # Your valid receiver.
 - name: some-other-receiver
   # ...
route:
 # default receiver
 receiver: some-other-receiver
 routes:
   # Child receiver.
   - matchers:
       - matchType: =
         name: alertname
         value: DeadMansSwitch
     receiver: blackhole
```

A detailed description of all parameters can be found in the official documentation.

Why can’t different scrape Intervals be set for individual targets?

The Prometheus developer Brian Brazil provides, probably, the most comprehensive answer to this question. In short, different scrapeIntervals are likely to cause the following complications:

- Increasing configuration complexity;
- Problems with writing queries and creating graphs;
- Short intervals are more like profiling an app, and Prometheus isn’t the best tool to do this in most cases.

The most appropriate value for scrapeInterval is in the range of 10-60s.

How do I limit Prometheus resource consumption?

To avoid situations when VPA requests more resources for Prometheus or Longterm Prometheus than those available on the corresponding node, you can explicitly limit VPA using module parameters:

- `vpa.longtermMaxCPU`
- `vpa.longtermMaxMemory`
- `vpa.maxCPU`
- `vpa.maxMemory`

How do I set up a ServiceMonitor or PodMonitor to work with Prometheus?

Add the `prometheus: main` label to the PodMonitor or ServiceMonitor. Add the label `prometheus.deckhouse.io/monitor-watcher-enabled: "true"` to the namespace where the PodMonitor or ServiceMonitor was created.

Example:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/monitor-watcher-enabled: "true"
---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: example-app
  namespace: frontend
  labels:
    prometheus: main
spec:
  selector:
    matchLabels:
      app: example-app
  endpoints:
    - port: web
```

How do I set up a Probe to work with Prometheus?

Add the `prometheus: main` to Probe. Add the label `prometheus.deckhouse.io/probe-watcher-enabled: "true"` to the namespace where the Probe was created.

Example:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/probe-watcher-enabled: "true"
---
apiVersion: monitoring.coreos.com/v1
kind: Probe
metadata:
  labels:
    app: prometheus
    component: probes
    prometheus: main
  name: cdn-is-up
  namespace: frontend
spec:
  interval: 30s
  jobName: httpGet
  module: http_2xx
  prober:
    path: /probe
    scheme: http
    url: blackbox-exporter.blackbox-exporter.svc.cluster.local:9115
  targets:
    staticConfig:
      static:
      - https://example.com/status
```

How do I set up a PrometheusRules to work with Prometheus?

Add the label `prometheus.deckhouse.io/rules-watcher-enabled: "true"` to the namespace where the PrometheusRules was created.

Example:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/rules-watcher-enabled: "true"
```

How to expand disk size

1. To request a larger volume for a PVC, edit the PVC object and specify a larger size in `spec.resources.requests.storage` field.
   - You can only expand a PVC if its storage class's `allowVolumeExpansion` field is set to true.
2. If storage doesn't support online resize, the message `Waiting for user to (re-)start a pod to finish file system resize of volume on node.` will appear in the PersistentVolumeClaim status.
3. Restart the Pod to complete the file system resizing.

How to get information about alerts in a cluster?

You can get information about active alerts not only in the Grafana/Prometheus web interface but in the CLI. This can be useful if you only have access to the cluster API server and there is no way to open the Grafana/Prometheus web interface.

Run the following command to get cluster alerts:

```
d8 k get clusteralerts
```

Example:

```
# d8 k get clusteralerts
NAME              ALERT                                      SEVERITY   AGE     LAST RECEIVED   STATUS
086551aeee5b5b24  ExtendedMonitoringDeprecatatedAnnotation   4          3h25m   38s             firing
226d35c886464d6e  ExtendedMonitoringDeprecatatedAnnotation   4          3h25m   38s             firing
235d4efba7df6af4  D8SnapshotControllerPodIsNotReady          8          5d4h    44s             firing
27464763f0aa857c  D8PrometheusOperatorPodIsNotReady          7          5d4h    43s             firing
ab17837fffa5e440  DeadMansSwitch                             4          5d4h    41s             firing
```

Run the following command to view a specific alert:

```
d8 k get clusteralerts <ALERT_NAME> -o yaml
```

Example:

```
# d8 k get clusteralerts 235d4efba7df6af4 -o yaml
alert:
  description: |
    The recommended course of action:
    1. Retrieve details of the Deployment: `kubectl -n d8-snapshot-controller describe deploy snapshot-controller`
    2. View the status of the Pod and try to figure out why it is not running: `kubectl -n d8-snapshot-controller describe pod -l app=snapshot-
controller`
  labels:
    pod: snapshot-controller-75bd776d76-xhb2c
    prometheus: deckhouse
    tier: cluster
  name: D8SnapshotControllerPodIsNotReady
  severityLevel: "8"
  summary: The snapshot-controller Pod is NOT Ready.
apiVersion: deckhouse.io/v1alpha1
kind: ClusterAlert
metadata:
  creationTimestamp: "2023-05-15T14:24:08Z"
  generation: 1
  labels:
    app: prometheus
    heritage: deckhouse
  name: 235d4efba7df6af4
  resourceVersion: "36262598"
  uid: 817f83e4-d01a-4572-8659-0c0a7b6ca9e7
status:
  alertStatus: firing
  lastUpdateTime: "2023-05-15T18:10:09Z"
  startsAt: "2023-05-10T13:43:09Z"
```

> The presence of a special alert `MissingDeadMansSwitch` in the cluster indicates problems in the performance of the monitoring components.

For the list of all available alerts in the Deckhouse Kubernetes Platform monitoring system, refer to the corresponding documentation page.

How do I add additional endpoints to a scrape config?

Add the label `prometheus.deckhouse.io/scrape-configs-watcher-enabled: "true"` to the namespace where the ScrapeConfig was created.

Example:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/scrape-configs-watcher-enabled: "true"
```

Add the ScrapeConfig with the required label `prometheus: main`:

```
apiVersion: monitoring.coreos.com/v1alpha1
kind: ScrapeConfig
metadata:
  name: example-scrape-config
  namespace: frontend
  labels:
    prometheus: main
spec:
  honorLabels: true
  staticConfigs:
    - targets: ['example-app.frontend.svc.{{ .Values.global.discovery.clusterDomain }}.:8080']
  relabelings:
    - regex: endpoint|namespace|pod|service
      action: labeldrop
    - targetLabel: scrape_endpoint
      replacement: main
    - targetLabel: job
      replacement: kube-state-metrics
  metricsPath: '/metrics'
```

The prometheus-metrics-adapter module

> i
> The module lifecycle stage: General Availability
> The module has requirements for installation

This module allows HPA and VPA autoscalers base their decisions on various metrics.

It installs an implementation of the Kubernetes resource metrics API, custom metrics API, and external metrics API to get Prometheus metrics.

As a result:

- `kubectl top` can collect Prometheus metrics via the adapter;
- autoscaling/v2 can be used for scaling applications (HPA);
- Prometheus data can be obtained using the Kubernetes API and utilized in other modules (Vertical Pod Autoscaler, etc.).

The following parameters serve as a basis for scaling:

- CPU (of the Pod),
- memory (of the Pod),
- rps (of the Ingress) — over 1,5,15 minutes (`rps Nm`),
- CPU (of the Pod) — over 1,5,15 minutes (`cpu Nm`) — average CPU utilization over N minutes,
- memory (of the Pod) — over 1,5,15 minutes (`memory_Nm`) — average Memory utilization over N minutes,
- any Prometheus metrics and any queries based on them.

How does it work?

This module registers `k8s-prometheus-adapter` as an external API service that extends the capabilities of the Kubernetes API. When some Kubernetes component (VPA, HPA) needs information about the resources consumed, it requests the Kubernetes API, which, in turn, proxies that request to the adapter. The adapter figures out (using its configuration file) how to calculate the metric and sends a request to Prometheus.

Third-party components

List of third-party software used in the `prometheus-metrics-adapter` module:

- Kubernetes Metrics Server

  License: Apache License 2.0

  A scalable, efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines.

Configuration

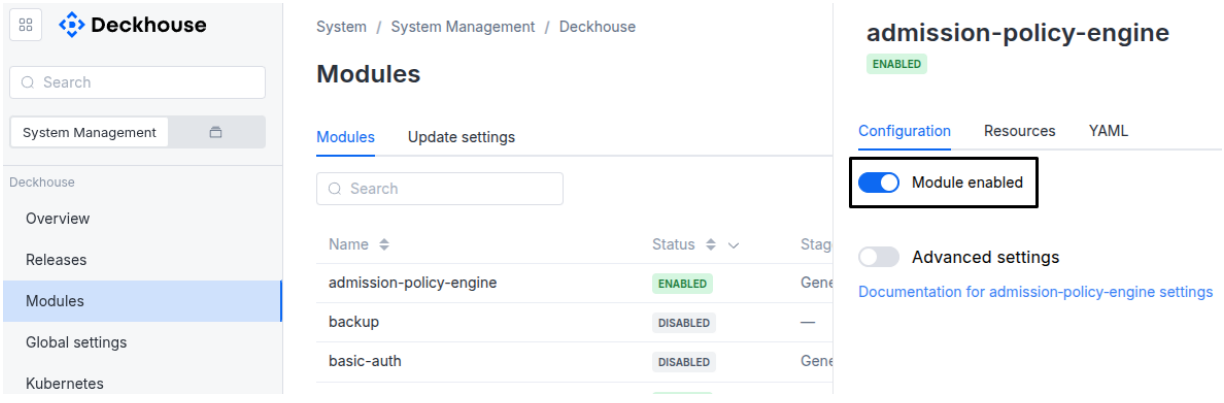
> i
> The module lifecycle stage: General Availability
> The module has requirements for installation

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `prometheus-metrics-adapter` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable prometheus-metrics-adapter
  ```

- **Using ModuleConfig `prometheus-metrics-adapter`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `prometheus-metrics-adapter` (create it if necessary);

  Example of a manifest to enable module `prometheus-metrics-adapter`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: prometheus-metrics-adapter
  spec:
    enabled: true
  ```

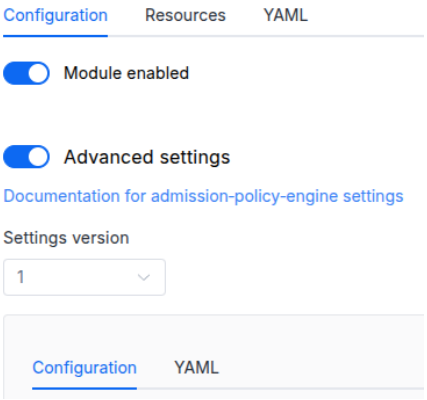
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `prometheus-metrics-adapter` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `prometheus-metrics-adapter` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `prometheus-metrics-adapter`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `prometheus-metrics-adapter` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc prometheus-metrics-adapter
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `prometheus-metrics-adapter` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `prometheus-metrics-adapter`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: prometheus-metrics-adapter
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the versions of other modules:

- `prometheus`: any version.

Generally, no configuration is required.

Parameters

- `highAvailability` — manually enable/disable the high availability mode. By default, this parameter is configured automatically (additional information about the HA mode for modules is available here).

Custom resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

ClusterDaemonSetMetric

Scope: Cluster

- v1beta1
- v1alpha1

ClusterDaemonSetMetric is handy interface for configuring custom DaemonSet-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

ClusterDaemonSetMetric is handy interface for configuring custom DaemonSet-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

ClusterDeploymentMetric

Scope: Cluster

- v1beta1
- v1alpha1

ClusterDeploymentMetric is handy interface for configuring custom Deployment-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

ClusterDeploymentMetric is handy interface for configuring custom Deployment-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

ClusterIngressMetric

Scope: Cluster

- v1beta1
- v1alpha1

ClusterIngressMetric is handy interface for configuring custom Ingress-related metrics in prometheus-metrics-adapter.

- **spec**
  object

Required value

- **spec.query**
  string

  Required value

  PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

ClusterIngressMetric is handy interface for configuring custom Ingress-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

ClusterPodMetric

Scope: Cluster

- v1beta1
- v1alpha1

ClusterPodMetric is handy interface for configuring custom Pod-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

Deprecated resource. Support for the resource might be removed in a later release.

ClusterPodMetric is handy interface for configuring custom Pod-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

ClusterServiceMetric

Scope: Cluster

- v1beta1
- v1alpha1

ClusterServiceMetric is handy interface for configuring custom Service-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

ClusterServiceMetric is handy interface for configuring custom Service-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

ClusterStatefulSetMetric

Scope: Cluster

- v1beta1
- v1alpha1

ClusterStatefulSetMetric is handy interface for configuring custom StatefulSet-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

ClusterStatefulSetMetric is handy interface for configuring custom StatefulSet-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

DaemonSetMetric

Scope: Namespaced

- v1beta1
- v1alpha1

DaemonSetMetric is handy interface for configuring custom DaemonSet-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

DaemonSetMetric is handy interface for configuring custom DaemonSet-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

DeploymentMetric

Scope: Namespaced

- v1beta1
- v1alpha1

DeploymentMetric is handy interface for configuring custom Deployment-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

DeploymentMetric is handy interface for configuring custom Deployment-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

    - **spec.query**
      string

      Required value

      PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

IngressMetric

Scope: Namespaced

- v1beta1
- v1alpha1

IngressMetric is handy interface for configuring custom Ingress-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

    - **spec.query**
      string

      Required value

      PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

IngressMetric is handy interface for configuring custom Ingress-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

    - **spec.query**
      string

      Required value

      PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

NamespaceMetric

Scope: Namespaced

- v1beta1
- v1alpha1

NamespaceMetric is handy interface for configuring custom Namespace-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

    - **spec.query**
      string

      Required value

      PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

NamespaceMetric is handy interface for configuring custom Namespace-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

    - **spec.query**
      string

      Required value

      PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

PodMetric

Scope: Namespaced

- v1beta1
- v1alpha1

PodMetric is handy interface for configuring custom Pod-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

    - **spec.query**
      string

      Required value

      PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use

keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

PodMetric is handy interface for configuring custom Pod-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

ServiceMetric

Scope: Namespaced

- v1beta1
- v1alpha1

ServiceMetric is handy interface for configuring custom Service-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

ServiceMetric is handy interface for configuring custom Service-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

StatefulSetMetric

Scope: Namespaced

- v1beta1
- v1alpha1

StatefulSetMetric is handy interface for configuring custom StatefulSet-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

StatefulSetMetric is handy interface for configuring custom StatefulSet-related metrics in prometheus-metrics-adapter.

- **spec**
  object

  Required value

  - **spec.query**
    string

    Required value

    PromQL-query which returns unambiguous value for your metric. Use grouping operators like sum() by(), max() by() etc. Also use keywords: «.LabelMatchers» with your optional applied comma-separated labels as label selector and «.GroupBy» as group-by-labels. Example: max(mymetric{«.LabelMatchers»,queue=”messages”}) by(«.GroupBy»)

Usage

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

Note that only HPA (Horizontal Pod Autoscaling) with apiVersion: autoscaling/v2, whose support has been available since Kubernetes v1.12, is discussed below.

Configuring HPA requires:

- defining what is being scaled (`.spec.scaleTargetRef`);
- defining the scaling range (`.spec.minReplicas` , `.scale.maxReplicas`);
- defining metrics to be used as the basis for scaling (`.spec.metrics`) and registering them with the Kubernetes API.

Metrics in terms of HPA are of three types:

- classic — of type (`.spec.metrics[].type`) “Resource”; these are used for simple scaling based on CPU and memory consumption;
- custom — of type (`.spec.metrics[].type`) “Pods” or “Object”;
- external — of type (`.spec.metrics[].type`) “External”.

Caution! By default, HPA uses different approaches for scaling:

- If the metrics indicate that scaling **up** is required, it is done immediately (`spec.behavior.scaleUp.stabilizationWindowSeconds` = 0). The only limitation is the rate of increase: pods can double in 15 seconds, but if there are less than 4 pods, 4 new pods will be added.
- If the metrics indicate that scaling **down** is required, it happens within 5 minutes (`spec.behavior.scaleUp.stabilizationWindowSeconds` = 300): suggestions for a new number of replicas are calculated, then the largest value is selected. There is no limit on the number of pods to be removed at once.

If metrics are subject to fluctuations that result in a surge of unnecessary application replicas, the following approaches are used:

- Wrapping the metric with an aggregation function (e. g., `avg_over_time()`) if the metric is defined by a PromQL query. For more details, see. example.
- Increasing the stabilization window (parameter `spec.behavior.scaleUp.stabilizationWindowSeconds`) in the *HorizontalPodAutoscaler* resource. During the this period, requests to increase the number of replicas will be accumulated, then the most modest request will be selected. This method is identical to applying the `min_over_time(<stabilizationWindowSeconds>)` aggregation function, but only if the metric is increasing and scaling **up** is required. For scaling **down**, the default settings usually work good enough. For more details, see example.
- Limiting the rate of increase of the new replica count with `spec.behavior.scaleUp.policies` .

Scaling types

The following metrics can be used to scale applications:

1. Classic metrics.
2. Custom namespace-scoped metrics. This type is suitable if you have a single application, the source of the metrics is in the namespace and it is tied to one of the objects.
3. Custom cluster-wide metrics. This type is suitable if you have many applications using the same metric, whose source is in the application namespace, and it is associated with one of the objects. Such metrics let you put common infrastructure components into a separate deployment (“infra”).
4. If the metric source is not tied to the application namespace, you can use external metrics. For example, metrics provided by a cloud provider or an external SaaS service.

Caution! We recommend using option 1 (classic metrics), or option 2 (custom metrics defined in the *Namespace*). In this case, we suggest defining the application configuration (including its autoscaling) in the app repository. You should consider options 3 and 4 only if you have a large collection of identical microservices.

Classic resource consumption-based scaling

Below is an example HPA configuration for scaling based on the classic metrics from `metrics.k8s.io` : CPU and memory utilization for pods. The `averageUtulization` value reflects the target percentage of resources that have been **requested**.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: app-hpa
  namespace: app-prod
spec:
  # Indicates the controller to be scaled (reference to a deployment or statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: app
  # Controller scaling limits.
  minReplicas: 1
  maxReplicas: 10
  # If the application is prone to short-term spikes in CPU consumption,
  # you can postpone the scaling decision to see if it is necessary.
  # By default, scaling up occurs immediately.
  behavior:
    scaleUp:
      stabilizationWindowSeconds: 300
  metrics:
  # CPU- and memory-based scaling.
  - type: Resource
    resource:
      name: cpu
      target:
        # Scaling will occur when the average CPU utilization of all pods in scaleTargetRef exceeds the specified value.
        # For a metric with type: Resource, only type: Utilization is available.
        type: Utilization
        # Scaling will occur if 1 core is requested for all Deployment pods and if over 700m is already in use on
average.
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        # Example of a scaling rule when the average memory usage of all pods in scaleTargetRef exceeds the given value.
        type: Utilization
        # Scaling will occur if 1 GB of memory is requested for the pods and more than 800 MB is already in use on
average.
        averageUtilization: 80
```

Scaling based on custom metrics

Registering custom metrics with Kubernetes API

Custom metrics must be registered with the `/apis/custom.metrics.k8s.io/` API, this registration is done by prometheus-metrics-adapter (and it also implements the API). The metrics can then be referenced using the *HorizontalPodAutoscaler* object. Customizing a vanilla prometheus-metrics-adapter is a time-consuming process. We made it easier by defining a set of Custom Resources with different Scopes:

- Namespaced:
  - `ServiceMetric`;
  - `IngressMetric`;
  - `PodMetric`;
  - `DeploymentMetric`;
  - `StatefulsetMetric`;
  - `NamespaceMetric`;
  - `DaemonSetMetric` (unavailable to users).
- Cluster:
  - `ClusterServiceMetric` (unavailable to users);
  - `ClusterIngressMetric` (unavailable to users);
  - `ClusterPodMetric` (unavailable to users);
  - `ClusterDeploymentMetric` (unavailable to users);
  - `ClusterStatefulsetMetric` (unavailable to users);
  - `ClusterDaemonSetMetric` (unavailable to users).

You can use the cluster-wide resource to define the metric globally, and use the *Namespace* to redefine it locally. Format is the same for all custom resources.

Using custom metrics in HPA

Once a custom metric is registered, it can be referenced. In terms of HPA, custom metrics can be of two types — `Pods` and `Object`.

`Object` refers to an object in the cluster that has metrics in Prometheus with corresponding labels (`namespace=XXX,ingress=YYYYY`). These labels will be substituted in place of `<<.LabelMatchers>>` in your custom request.

```
apiVersion: deckhouse.io/v1beta1
kind: IngressMetric
metadata:
  name: mymetric
  namespace: mynamespace
spec:
  query: sum(rate(ingress_nginx_detail_requests_total{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>) OR on() vector(0)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # Specifies the controller to be scaled (reference to a deployment or statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 1
  maxReplicas: 2
  # Metrics to use for scaling.
  # An example of using custom metrics.
  metrics:
  - type: Object
    object:
      # An object that has metrics in Prometheus.
      describedObject:
        apiVersion: networking.k8s.io/v1
        kind: Ingress
        name: myingress
      metric:
        # A metric registered with the IngressMetric or ClusterIngressMetric custom resource.
        # You can use rps_1m, rps_5m, or rps_15m that come with the prometheus-metrics-adapter module.
        name: mymetric
      target:
        # You can use `Value` or `AverageValue` for metrics of type Object.
        type: AverageValue
        # Scaling occurs if the average value of the custom metric for all pods in the Deployment deviates significantly
from 10.
        averageValue: 10
```

`Pods` — all pods will be selected from the resource managed by HPA and metrics will be collected for each pod with the relevant labels (`namespace=XXX`, `pod=YYYY-sadiq`, `namespace=XXX`, `pod=YYYY-e3adf`, etc.). Then the HPA will calculate an average value based on these metrics and will use it for scaling.

Using custom metrics with the RabbitMQ queue size

In the example below, scaling is performed based on the `send_forum_message` queue in RabbitMQ for which the `rmq` service is registered. If the number of messages in this queue exceeds 42, scaling is carried out.

```
apiVersion: deckhouse.io/v1beta1
kind: ServiceMetric
metadata:
  name: rmq-queue-forum-messages
  namespace: mynamespace
spec:
  query: sum (rabbitmq_queue_messages{<<.LabelMatchers>>,queue=~"send_forum_message",vhost="/"}) by (<<.GroupBy>>)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # The controller to be scaled (reference to a deployment or statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myconsumer
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Object
    object:
      describedObject:
        apiVersion: v1
        kind: Service
        name: rmq
      metric:
        name: rmq-queue-forum-messages
      target:
        type: Value
        value: 42
```

Using volatile custom metrics

This example improves on the previous one.

In the example below, scaling is based on the `send_forum_message` queue in RabbitMQ, for which the `rmq` service is registered. If the number of messages in this queue exceeds 42, scaling takes place. The MQL function `avg_over_time()` smooths (averages the metric) to avoid over-scaling due to short-term spikes in the number of messages.

```
apiVersion: deckhouse.io/v1beta1
kind: ServiceMetric
metadata:
  name: rmq-queue-forum-messages
  namespace: mynamespace
spec:
  query: sum (avg_over_time(rabbitmq_queue_messages{<<.LabelMatchers>>,queue=~"send_forum_message",vhost="/"}[5m])) by
(<<.GroupBy>>)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # The controller to be scaled (reference to a deployment or statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myconsumer
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Object
    object:
      describedObject:
        apiVersion: v1
        kind: Service
        name: rmq
      metric:
        name: rmq-queue-forum-messages
      target:
        type: Value
        value: 42
```

Examples involving custom metrics of type `Pods`

In the example below, the number of workers is scaled based on the percentage of active php-fpm workers. The trigger is the average number of php-fpm-workers in `mybackend` *Deployment*, which should not exceed 5.

```
apiVersion: deckhouse.io/v1beta1
kind: PodMetric
metadata:
  name: php-fpm-active-workers
spec:
  query: sum (phpfpm_processes_total{state="active",<<.LabelMatchers>>}) by (<<.GroupBy>>)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # The controller to be scaled (reference to a deployment or statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: mybackend
  minReplicas: 1
  maxReplicas: 5
  metrics:
  # HPA has to loop through all the Deployment pods and collect metrics from them.
  - type: Pods
    # Unlike type: Object, you don't have to specify describedObject.
    pods:
      metric:
        # A custom metric registered using the PodMetric custom resource.
        name: php-fpm-active-workers
      target:
        # For metrics of type: Pods, only AverageValue can be used.
        type: AverageValue
        # The scaling will take place if the average metric value of all Deployment pods exceeds 5.
        averageValue: 5
```

Scaling the Deployment based on the percentage number of active php-fpm-workers:

```
---
apiVersion: deckhouse.io/v1beta1
kind: PodMetric
metadata:
  name: php-fpm-active-worker
spec:
  # Percentage of active php-fpm-workers. The round() function gets rid of millipercentages in HPA.
  query: round(sum by(<<.GroupBy>>) (phpfpm_processes_total{state="active",<<.LabelMatchers>>}) / sum by(<<.GroupBy>>)
(phpfpm_processes_total{<<.LabelMatchers>>}) * 100)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: {{ .Chart.Name }}-hpa
spec:
  # The cpntroller to be scaled (reference to a deployment or statefulset).
  scaleTargetRef:
    apiVersion: apps/v1beta1
    kind: Deployment
    name: {{ .Chart.Name }}
  minReplicas: 4
  maxReplicas: 8
  metrics:
  - type: Pods
    pods:
      metric:
        name: php-fpm-active-worker
      target:
        type: AverageValue
        # Scaling will take place if, on average, the Deployment has 80% of the workers in use.
        averageValue: 80
```

Registering external metrics with the Kubernetes API

The `prometheus-metrics-adapter` module supports the `externalRules` mechanism. It allows you to define custom PromQL queries and register them as metrics.

A universal rule that allows you to create your own metrics without customization in `prometheus-metrics-adapter` has been added in the installation examples — “any metric in Prometheus with the name `kube_adapter_metric_<name>` will be registered in the API under the name `<name>`”. Then, you just need to write an exporter that will export such a metric or create a recording rule in Prometheus that will aggregate your metric based on other metrics.

Below is an example of *CustomPrometheusRules*:

The example showcases Prometheus custom rules for the `mymetric` metric.

```
apiVersion: deckhouse.io/v1
kind: CustomPrometheusRules
metadata:
  # The recommended template for naming your CustomPrometheusRules.
  name: prometheus-metrics-adapter-mymetric
spec:
  groups:
  # The recommended template
  - name: prometheus-metrics-adapter.mymetric
    rules:
    # The name of your new metric
    # Note that the 'kube_adapter_metric_' prefix is mandatory.
    - record: kube_adapter_metric_mymetric
      # A request with results that will end up in the final metric; there's no point in attaching extra labels to it.
      expr: sum(ingress_nginx_detail_sent_bytes_sum) by (namespace,ingress)
```

Using external metrics in HPA

Once an external metric is registered, you can refer to it.

```
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # The controller to be scaled (reference to a deployment or statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 1
  maxReplicas: 2
  metrics:
  # Scaling based on the external metrics
  - type: External
    external:
      metric:
        # The metric we registered by creating a metric in Prometheus (kube_adapter_metric_mymetric), but without the
prefix 'kube_adapter_metric_'.
        name: mymetric
        selector:
          # For external metrics, you can and should refine the request with labels.
          matchLabels:
            namespace: mynamespace
            ingress: myingress
      target:
        # Only `type: Value` can be used for metrics of type External.
        type: Value
        # Scaling will take place if the value of our metric exceeds 10.
        value: 10
```

Using the queue size in Amazon SQS

To install an exporter to integrate with SQS:

1. Create a dedicated “service” Git repository (alternatively, you could use, e.g., an “infrastructure” repository).
2. Copy the exporter installation and the script to it — these will be used to create the necessary *CustomPrometheusRules*.

That’s it, you have integrated the cluster. In case you need to configure autoscaling for just one application (in a single namespace), we recommend installing the exporter together with that application and using `NamespaceMetrics`.

The following is an example of an exporter (e. g., sqs-exporter) to retrieve metrics from Amazon SQS if:

- a `send_forum_message` queue is running in Amazon SQS;
- scaling is done when the number of messages in that queue exceeds 42.

```
apiVersion: deckhouse.io/v1
kind: CustomPrometheusRules
metadata:
  # The recommended name — prometheus-metrics-adapter-<metric name>.
  name: prometheus-metrics-adapter-sqs-messages-visible
  # Pay attention!
  namespace: d8-monitoring
  labels:
    # Pay attention!
    prometheus: main
    # Pay attention!
    component: rules
spec:
  groups:
  - name: prometheus-metrics-adapter.sqs_messages_visible # the recommended template
    rules:
    - record: kube_adapter_metric_sqs_messages_visible # Pay attention! The 'kube_adapter_metric_' prefix is required.
      expr: sum (sqs_messages_visible) by (queue)
---
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: myhpa
  namespace: mynamespace
spec:
  # The targets of scaling (link to a deployment or statefulset).
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myconsumer
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: External
    external:
      metric:
        # Must match CustomPrometheusRules record name without 'kube_adapter_metric_' prefix.
        name: sqs_messages_visible
        selector:
          matchLabels:
            queue: send_forum_messages
      target:
        type: Value
        value: 42
```

Debugging

How do I get a list of custom metrics?

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/
```

How do I get the value of a metric associated with an object?

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-namespace/services/*/my-service-metric
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-namespace/ingresses/*/rps_1m
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-namespace/ingresses/*/mymetric
```

How do I get the value of a metric created via `NamespaceMetric`?

```
d8 k get --raw /apis/custom.metrics.k8s.io/v1beta1/namespaces/my-namespace/metrics/my-ns-metric
```

How do I get external metrics?

```
d8 k get --raw /apis/external.metrics.k8s.io/v1beta1
d8 k get --raw /apis/external.metrics.k8s.io/v1beta1/namespaces/d8-ingress-nginx/d8_ingress_nginx_ds_cpu_utilization
```

The Prometheus Pushgateway module

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

This module installs Prometheus Pushgateway into the cluster. It gets metrics from the app and pushes them to Prometheus.

Additional information:

- When to use `Prometheus Pushgateway`.
- How to collect metrics using `Prometheus Pushgateway`.

Third-party components

List of third-party software used in the `prometheus-pushgateway` module:

- Prometheus Pushgateway

  License: Apache License 2.0

  A tool to allow ephemeral and batch jobs to expose their metrics to Prometheus.

Configuration

> The module lifecycle stage: General Availability
> The module has requirements for installation

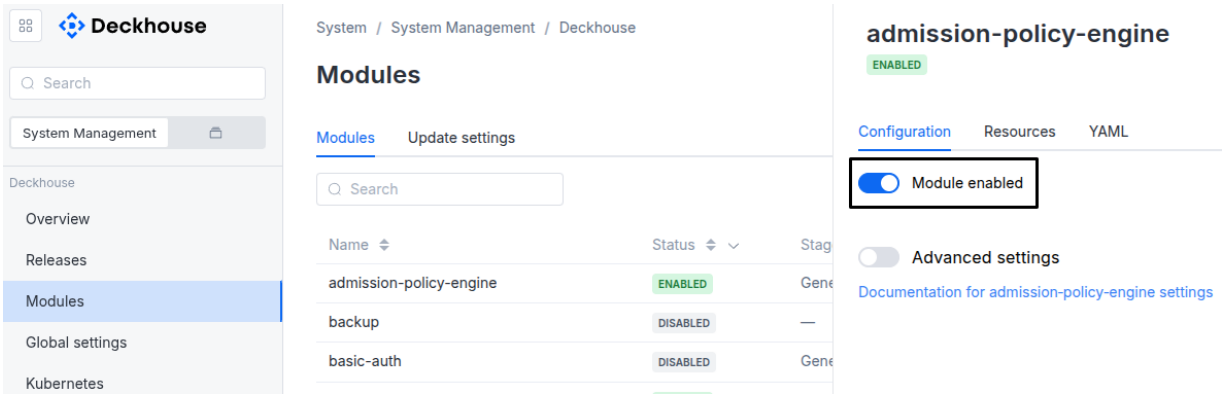
This module installs Prometheus Pushgateway into the cluster. It receives metrics from the app and pushes them to Prometheus.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `prometheus-pushgateway` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable prometheus-pushgateway
```

- **Using ModuleConfig `prometheus-pushgateway`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `prometheus-pushgateway` (create it if necessary);

  Example of a manifest to enable module `prometheus-pushgateway`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus-pushgateway
spec:
  enabled: true
```

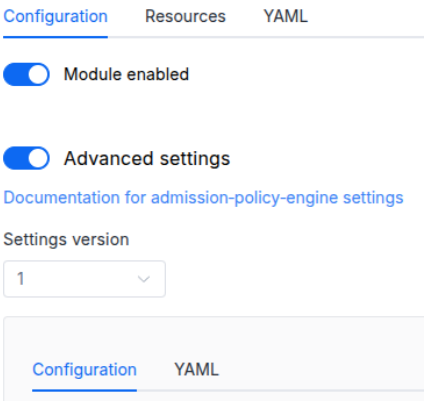
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `prometheus-pushgateway` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `prometheus-pushgateway` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `prometheus-pushgateway`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `prometheus-pushgateway` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc prometheus-pushgateway
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `prometheus-pushgateway` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `prometheus-pushgateway`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: prometheus-pushgateway
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the versions of other modules:

- `prometheus`: any version.

Parameters

Schema version: 1

- **settings**
  object
  - **settings.instances**
    array of strings

    Required value

    This parameter contains a list of instances; a separate PushGateway will be created for each instance.

    Example:

    ```
    instances:
    - first
    - second
    ```
  - **settings.nodeSelector**
    object

    The same as in the pods’ `spec.nodeSelector` parameter in Kubernetes.

    If the parameter is omitted or `false`, it will be determined automatically.
  - **settings.tolerations**
    array of objects

    The same as in the pods’ `spec.tolerations` parameter in Kubernetes;

    If the parameter is omitted or `false`, it will be determined automatically.

    - **settings.tolerations.effect**
      string
    - **settings.tolerations.key**
      string
    - **settings.tolerations.operator**
      string
    - **settings.tolerations.tolerationSeconds**
      integer
    - **settings.tolerations.value**
      string

Examples

> **i**
> The module lifecycle stage: General Availability
> The module has requirements for installation

Example of the module configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus-pushgateway
spec:
  version: 1
  enabled: true
  settings:
    instances:
    - first
    - second
    - another
```

PushGateway address (from a container pod): `http://first.kube-prometheus-pushgateway:9091`.

Pushing a metric

An example of pushing a metric using curl:

```
echo "test_metric{env="dev"} 3.14" | curl --data-binary @- http://first.kube-prometheus-pushgateway:9091/metrics/job/myapp
```

The metrics will be available in Prometheus in 30 seconds (after the data are scraped). An example:

```
test_metric{container="prometheus-pushgateway", env="dev", exported_job="myapp",
    instance="10.244.1.155:9091", job="prometheus-pushgateway", pushgateway="prometheus-pushgateway", tier="cluster"} 3.14
```

i The job name (`myapp` in the example) will be available in Prometheus in the label `exported_job`, and not `job` (because the label `job` already exists in Prometheus and is renamed when receiving metrics from PushGateway).

i You may need to get a list of all available job names to choose a unique name (in order not to spoil existing graphs and alerts). Use the following query to get a list of all existing jobs: `count({__name__=~".+"}) by (job)`.

Deleting metrics

An example of deleting all metrics of a group `{instance="10.244.1.155:9091",job="myapp"}` using curl:

```
curl -X DELETE http://first.kube-prometheus-pushgateway:9091/metrics/job/myapp/instance/10.244.1.155:9091
```

Since PushGateway stores the scraped metrics in memory, **all metrics will be lost when the Pod is restarted**.

Registry Module

⚠ The module lifecycle stage: Preview

Description

The module manages the registry settings for DKP components.

The module can operate in the following modes:

- `Direct`: Provides direct access to an external registry via the fixed address `registry.d8-system.svc:5001/system/deckhouse`. This fixed address prevents Deckhouse images from being re-downloaded and components from being restarted when registry parameters are changed. Switching between modes and registries is done through the `deckhouse` ModuleConfig. The switching process is automatic — see the usage examples for more information.
- `Proxy`: Using an internal caching proxy registry that accesses an external registry, with the caching proxy registry running on control-plane (master) nodes. This mode reduces the number of requests to the external registry by caching images. Cached data is stored on the control-plane (master) nodes. Access to the internal registry is via the fixed address `registry.d8-system.svc:5001/system/deckhouse`, similar to the `Direct` mode. Switching between modes and registries is done through the `deckhouse` ModuleConfig. The switching process is automatic — see the usage examples for more information.
- `Local`: Using a local internal registry, with the registry running on control-plane (master) nodes. This mode allows the cluster to operate in an isolated environment. All data is stored on the control-plane (master) nodes. Access to the internal registry is via the fixed address `registry.d8-system.svc:5001/system/deckhouse`, similar to the `Direct` and `Proxy` modes. Switching between modes and registries is done through the `deckhouse` ModuleConfig. The switching process is automatic — see the usage examples for more information.
- `Unmanaged`: Operation without using the internal registry. Access within the cluster is performed directly to the external registry. There are two types of the `Unmanaged` mode:
  - Configurable: A mode managed via the `registry` module. Switching between modes and registries is handled through the `deckhouse` ModuleConfig. The switch is performed automatically (see usage examples for details).
  - Non-configurable (deprecated): The default mode. Configuration parameters are set during cluster installation or changed in a running cluster using the (deprecated) `helper change registry` command.

Restrictions and features of using the module

The `registry` module has a number of limitations and features related to installation, operating conditions, and mode switching.

Cluster installation limitations

- DKP cluster bootstrap is supported only in `Direct`, `Unmanaged`, and `Proxy` modes (bootstrap in `Local` mode is not supported). Registry settings during cluster installation are configured via the `deckhouse` ModuleConfig.
- To launch a cluster in the non-configurable `Unmanaged` mode (Legacy), registry parameters must be specified in `initConfiguration`.

Operating conditions restrictions

The module works under the following conditions:

- If CRI containerd or containerd v2 is used on the cluster nodes. To configure CRI, refer to the `ClusterConfiguration` configuration.
- The cluster is fully managed by DKP. The module will not work in Managed Kubernetes clusters.
- The `Local` and `Proxy` modes are only supported on static clusters.

Mode switching restrictions

Mode switching restrictions are as follows:

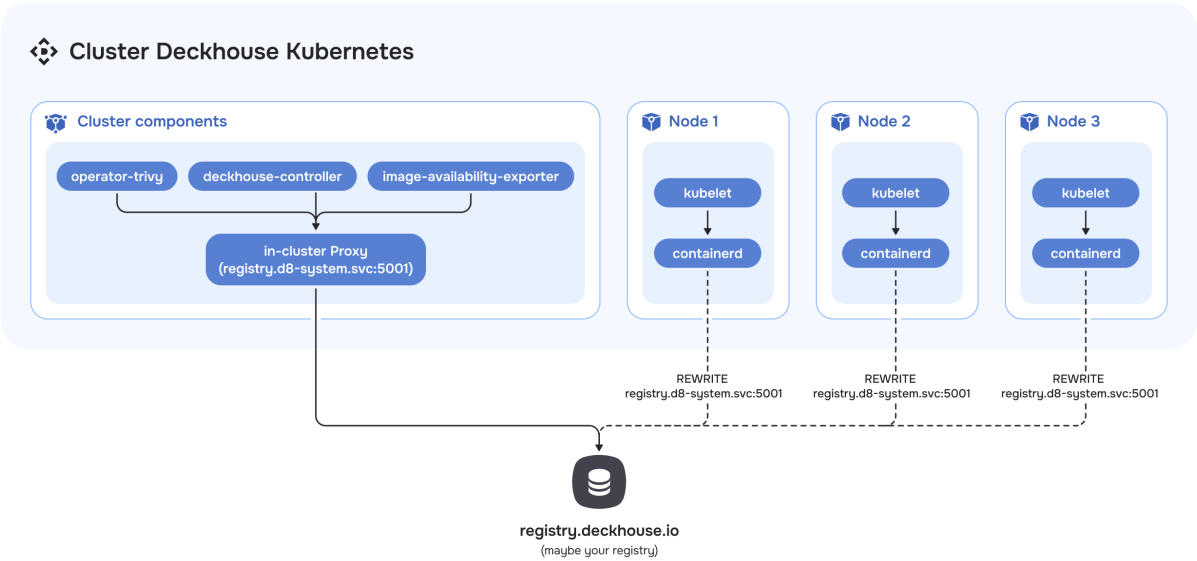
- Changing registry parameters and switching modes is only available after the bootstrap phase is fully complete.
- For the first switch, migration of user registry configurations must be performed. For more details, see the Registry Module: FAQ section.
- Switching to the non-configurable `Unmanaged` mode is only available from the `Unmanaged` mode. For more details, see the Registry Module: FAQ section.
- Switching between `Local` and `Proxy` modes is only possible via the intermediate `Direct` or `Unmanaged` modes. Example switching sequence: `Local/Proxy` → `Direct` → `Proxy/Local`.

Direct Mode Architecture

In Direct mode, registry requests are processed directly, without intermediate caching.

CRI requests to the registry are redirected based on its configuration, which is defined in the `containerd` configuration.

For components such as `operator-trivy`, `image-availability-exporter`, `deckhouse-controller`, and others that access the registry directly, requests will go through the in-cluster proxy located on the master nodes.



Proxy Mode Architecture

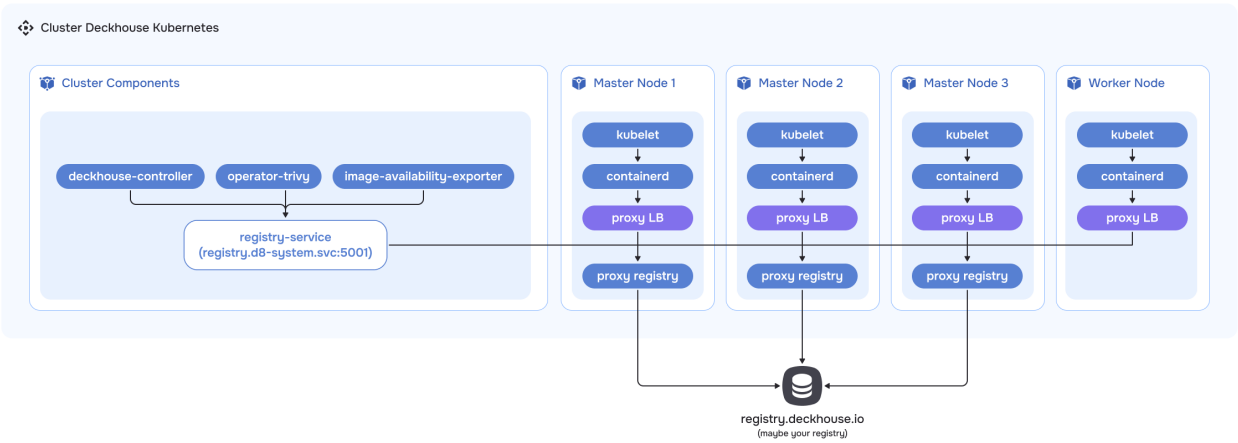
> ⚠ It is recommended to use separate disks for storing registry (`/opt/deckhouse/registry`) and etcd data. Using a single disk may lead to etcd performance degradation during registry operations.

The `Proxy` mode allows the registry to act as an intermediate proxy server between the client and the remote registry.

The caching proxy registry is launched as static pods on control-plane (master) nodes. Cached data is stored on the control-plane (master) nodes in the `/opt/deckhouse/registry` directory.

To ensure high availability of the caching proxy registry, a load balancer is deployed on each cluster node. Access to the proxy registry from CRI goes through this load balancer. The configuration for accessing the load balancer is set in the `containerd` configuration.

For components such as `operator-trivy`, `image-availability-exporter`, `deckhouse-controller`, and others that access the registry directly, requests will go through the caching proxy registry.



Local Mode Architecture

⚠ It is recommended to use separate disks for storing registry (`/opt/deckhouse/registry`) and etcd data. Using a single disk may lead to etcd performance degradation during registry operations.

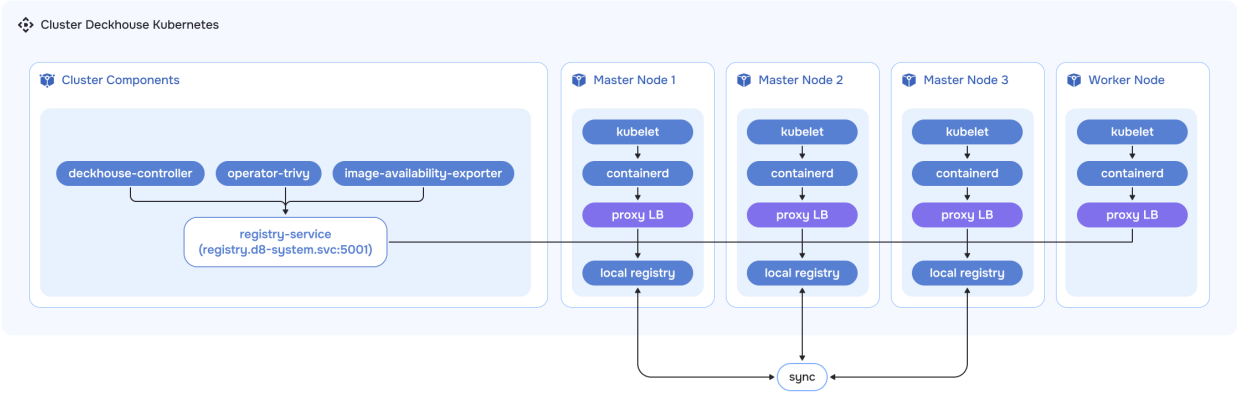
The `Local` mode allows creating a local copy of the registry inside the cluster. Images from the remote registry are fully copied to local storage and synchronized between replicas of the local registry.

The operation of the local registry is identical to that of the caching proxy registry. The local registry is launched as static pods on control-plane (master) nodes. Registry data is stored on the control-plane (master) nodes in the `/opt/deckhouse/registry` directory.

To ensure high availability of the local registry, a load balancer is deployed on each cluster node. Access to the local registry from CRI goes through this load balancer. The configuration for accessing the load balancer is set in the `containerd` configuration.

For components such as `operator-trivy`, `image-availability-exporter`, `deckhouse-controller`, and others that access the registry directly, requests will go to the local registry.

The local registry is populated using the `d8` tool with the `d8 mirror push/pull` commands. For more details, see the «Registry Module: Usage Examples» section.



Module registry: configuration

⚠ The module lifecycle stage: Preview

⚠ To configure connection parameters to the container registry and manage the container registry operating mode, use the `registry` section of the `deckhouse` module configuration.

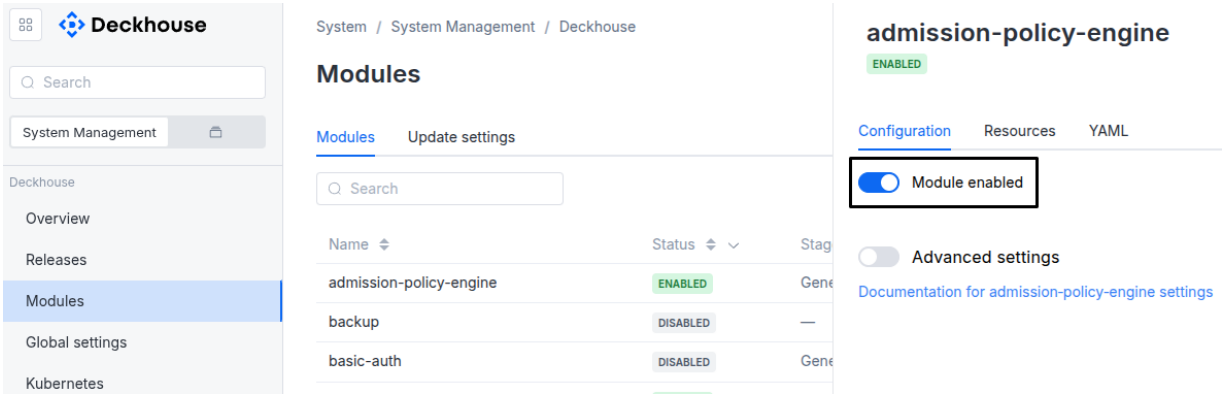
The parameters of the `registry` module itself are specified in ModuleConfig `registry`.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `registry` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable registry
```

- **Using ModuleConfig `registry`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `registry` (create it if necessary);

  Example of a manifest to enable module `registry`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry
spec:
  enabled: true
```

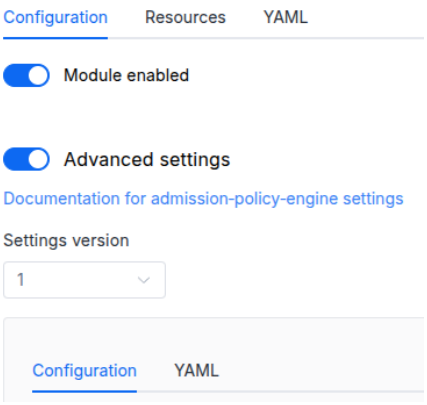
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `registry` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `registry` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `registry`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `registry` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc registry
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `registry` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `registry`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: registry
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.https**
    object

    What certificate type to use.

    This parameter completely overrides the `global.modules.https` settings.

    Examples:

    ```
    https:
      mode: Disabled
    ```

    ```
    https:
      mode: OnlyInURI
    ```

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

- **settings.https.certManager**
  object

  Parameters for certmanager.

  - **settings.https.certManager.clusterIssuerName**
    string

    What ClusterIssuer to use for getting an SSL certificate (currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available; also, you can define your own).

    Default: `letsencrypt`

    Example:

    ```
    clusterIssuerName: letsencrypt
    ```

- **settings.https.customCertificate**
  object

  Parameters for custom certificate usage.

  - **settings.https.customCertificate.secretName**
    string

    The name of the secret in the `d8-system` namespace to use with the registry ingress.

    This secret must have the kubernetes.io/tls format.

- **settings.https.mode**
  string

  The HTTPS usage mode:

  - `CertManager`: The registry ingress is accessed over HTTPS using a certificate obtained from a clusterIssuer specified in the `certManager.clusterIssuerName` parameter.
  - `CustomCertificate`: The registry ingress is accessed over HTTPS using a certificate from the `d8-system` namespace.
  - `Disabled`: In this mode, the registry ingress can only be accessed over HTTP.
  - `OnlyInURI`: The registry ingress will work over HTTP (thinking that there is an external HTTPS load balancer in front of it that terminates HTTPS traffic). Load balancer should provide a redirect from HTTP to HTTPS.

  Default: `CertManager`

  Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller used for the registry.

  Optional. By default, the `modules.ingressClass` global value is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.whitelistSourceRanges**
  array of strings

  A list of CIDR-formatted addresses allowed to connect to the registry. If not specified, connections from any address are allowed.

  Example:

  ```
  whitelistSourceRanges:
  - 10.0.0.0/10
  - 192.168.0.0/16
  ```

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))?$`

FAQ

⚠ The module lifecycle stage: Preview

How to Migrate to the registry module?

During the migration, Containerd v1 will switch to the new registry configuration format. Containerd v2 uses the new format by default. For more details, see the section with a description of configuration methods.

For containerd v2

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

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

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

Specify this configuration when setting up the `Unmanaged` mode:

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

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

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

For Containerd v1

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

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

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

    REGISTRY_URL=private.registry.example

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

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

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

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

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

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

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

   Specify this configuration when setting up the `Unmanaged` mode:

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

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

   Example output:

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

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

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

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

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

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

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

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

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

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

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

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

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

```
d8 k get nodegroupconfiguration
```

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

How to Migrate Back from the Registry Module?

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

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

   Example configuration:

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

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

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

3. Switch the registry to the non-configurable `Unmanaged` mode. Example configuration:

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

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

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

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

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

    REGISTRY_URL=private.registry.example

    mkdir -p /etc/containerd/conf.d
    bb-sync-file /etc/containerd/conf.d/additional_registry.toml - << EOF
    [plugins]
      [plugins."io.containerd.grpc.v1.cri"]
        [plugins."io.containerd.grpc.v1.cri".registry]
          [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
            [plugins."io.containerd.grpc.v1.cri".registry.mirrors."${REGISTRY_URL}"]
              endpoint = ["https://${REGISTRY_URL}"]
          [plugins."io.containerd.grpc.v1.cri".registry.configs]
            [plugins."io.containerd.grpc.v1.cri".registry.configs."${REGISTRY_URL}".auth]
              username = "username"
              password = "password"
              # OR
              auth = "dXNlcm5hbWU6cGFzc3dvcmQ="
    EOF
```

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

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

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

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

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

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

How to check the registry mode switch status?

The status of the registry mode switch can be retrieved using the following command:

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

Example output:

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

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

Description of conditions:

Condition	Description
`ContainerdConfigPreflightReady`	State of the containerd configuration preflight check. Verifies there are no custom containerd auth configurations on the nodes.
`TransitionContainerdConfigReady`	State of preparing the containerd configuration for the new mode. Verifies that the configuration contains both the old and new mode settings.

Condition	Description
`FinalContainerdConfigReady`	State of finalizing the switch to the new containerd mode. Verifies that the containerd configuration has been successfully applied and contains only the new mode settings.
`DeckhouseRegistrySwitchReady`	State of switching Deckhouse and its components to use the new registry. `True` means Deckhouse successfully switched and is ready to operate.
`InClusterProxyReady`	State of In-Cluster Proxy readiness. Checks that the In-Cluster Proxy has started successfully and is running.
`CleanupInClusterProxy`	State of cleaning up the In-Cluster Proxy if it is not needed in the selected mode. Verifies that all related resources have been removed.
`NodeServicesReady`	State of Node Services Manager and Static-Pod registry readiness. Verifies that the Node Services Manager is successfully launched and operational, and that the Static-Pod registry has been successfully deployed by it.
`CleanupNodeServices`	State of cleaning up the Node Services Manager and Static-Pod registry if they are not needed in the selected mode. Verifies that all related resources have been removed.
`RegistryContainsRequiredImages`	State of checking the registry for the presence of required images.
`Ready`	Overall state of registry readiness in the selected mode. Indicates that all other conditions are met and the `module` is ready to operate.

Module registry: usage example

⚠ The module lifecycle stage: Preview

⚠ If, during the switching process, the image of a module did not reload and the module did not reinstall, use the instructions to resolve the issue.

Switching to the `Direct` Mode

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

✖ The first switch from `Unmanaged` to `Direct` mode will result in a full restart of all DKP components.

1. Before switching, perform the migration to use the `registry` module.
2. Make sure the `registry` module is enabled and running. To do this, execute the following command:

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

   Example output:

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

3. Make sure all master nodes are in the `Ready` state and do not have the `SchedulingDisabled` status, using the following command:

   ```
   d8 k get nodes
   ```

   Example output:

   ```
   NAME       STATUS   ROLES                ...
   master-0   Ready    control-plane,master  ...
   master-1   Ready    control-plane,master  ...
   master-2   Ready    control-plane,master  ...
   ```

   Example of output when the master node (`master-2` in the example) is in the `SchedulingDisabled` status:

   ```
   NAME       STATUS                     ROLES                ...
   master-0   Ready    control-plane,master  ...
   master-1   Ready    control-plane,master  ...
   master-2   Ready,SchedulingDisabled    control-plane,master  ...
   ```

4. Ensure the Deckhouse job queue is empty and contains no errors:

   ```
   d8 system queue list
   ```

   Example output:

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

5. Set the `Direct` mode configuration in the ModuleConfig `deckhouse`. If you're using a registry other than `registry.deckhouse.io`, refer to the `deckhouse` module documentation for correct configuration.

Configuration example:

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

6. Check the registry switch status in the `registry-state` secret using this guide.

   Example output:

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

Switching to the `Proxy` Mode

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

- The first switch from `Unmanaged` to `Proxy` mode will result in a full restart of all DKP components.
- Switching from `Local` mode to `Proxy` mode is not available. To switch from `Local` mode, you must switch the registry to another available mode (for example: `Direct`).

1. Before switching, perform the migration to use the `registry` module.
2. Make sure the `registry` module is enabled and running. To do this, execute the following command:

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

   Example output:

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

3. Make sure all master nodes are in the `Ready` state and do not have the `SchedulingDisabled` status, using the following command:

```
d8 k get nodes
```

   Example output:

```
NAME       STATUS   ROLES                ...
master-0   Ready    control-plane,master  ...
master-1   Ready    control-plane,master  ...
master-2   Ready    control-plane,master  ...
```

   Example of output when the master node (`master-2` in the example) is in the `SchedulingDisabled` status:

```
NAME       STATUS                     ROLES                ...
master-0   Ready    control-plane,master  ...
master-1   Ready    control-plane,master  ...
master-2   Ready,SchedulingDisabled    control-plane,master  ...
```

4. Ensure the Deckhouse job queue is empty and contains no errors:

```
d8 system queue list
```

   Example output:

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

5. Set the `Proxy` mode configuration in the ModuleConfig `deckhouse`. If you're using a registry other than `registry.deckhouse.io`, refer to the `deckhouse` module documentation for correct configuration.

   Configuration example:

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

6. Check the registry switch status in the `registry-state` secret using this guide.

   Example output:

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

Switching to the `Local` Mode

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

- The first switch from `Unmanaged` to `Local` mode will result in a full restart of all DKP components.
- Switching from `Proxy` mode to `Local` mode is not available. To switch from `Proxy` mode, you must switch the registry to another available mode (for example: `Direct`).

1. Before switching, perform the migration to use the `registry` module.
2. Make sure the `registry` module is enabled and running. To do this, execute the following command:

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

   Example output:

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

3. Make sure all master nodes are in the `Ready` state and do not have the `SchedulingDisabled` status, using the following command:

```
d8 k get nodes
```

   Example output:

```
NAME       STATUS   ROLES                ...
master-0   Ready    control-plane,master  ...
master-1   Ready    control-plane,master  ...
master-2   Ready    control-plane,master  ...
```

   Example of output when the master node (`master-2` in the example) is in the `SchedulingDisabled` status:

```
NAME       STATUS                     ROLES                ...
master-0   Ready    control-plane,master  ...
master-1   Ready    control-plane,master  ...
master-2   Ready,SchedulingDisabled    control-plane,master  ...
```

4. Ensure the Deckhouse job queue is empty and contains no errors:

```
d8 system queue list
```

   Example output:

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

5. Prepare archives with DKP images of the current version. To do this, use the `d8 mirror` command.

   Example:

```
TAG=$(
 d8 k -n d8-system get deployment/deckhouse -o yaml \
 | yq -r '.spec.template.spec.containers[] | select(.name == "deckhouse").image | split(":")[-1]'
) && echo "TAG: $TAG"

EDITION=$(
 d8 k -n d8-system exec -it svc/deckhouse-leader -- deckhouse-controller global values -o yaml \
 | yq .deckhouseEdition
) && echo "EDITION: $EDITION"
```

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

6. Set the `Local` mode configuration in the ModuleConfig `deckhouse`.

   Configuration example:

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

7. Check the registry switch status in the `registry-state` secret using this guide. In the status, you need to wait for the `RegistryContainsRequiredImages` check to start. The condition will show the absence or presence of images in the running local registry.

   Example output:

```
conditions:
# ...
- lastTransitionTime: "..."
  message: |-
    Mode: Default
    master-1: 0 of 166 items processed, 166 items with errors:
    - source: module/control-plane-manager/control-plane-manager133
      image:
10.128.0.5:5001/system/deckhouse@sha256:00202db19b40930f764edab5695f450cf709d50736e012055393447b3379414a
      error: HEAD
https://10.128.0.5:5001/v2/system/deckhouse/manifests/sha256:00202db19b40930f764edab5695f450cf709d50736e012055393
447b3379414a: unexpected status code 404 Not Found (HEAD responses have no body, use GET for details)
    - source: module/cloud-provider-yandex/cloud-metrics-exporter
      image:
10.128.0.5:5001/system/deckhouse@sha256:05517a86fcf0ec4a62d14ed7dc4f9ffd91c05716b8b0e28263da59edf11f0fad
      error: HEAD
https://10.128.0.5:5001/v2/system/deckhouse/manifests/sha256:05517a86fcf0ec4a62d14ed7dc4f9ffd91c05716b8b0ed86d6a1
f465f4556fb8: unexpected status code 404 Not Found (HEAD responses have no body, use GET for details)
    - source: module/control-plane-manager/kube-controller-manager132
      image:
10.128.0.5:5001/system/deckhouse@sha256:13f24cc717698682267ed2b428e7399b145a4d8ffe96ad1b7a0b3269b17c7e61
      error: HEAD
https://10.128.0.5:5001/v2/system/deckhouse/manifests/sha256:13f24cc717698682267ed2b428e7399b145a4d8ffe96ad1b7a0b
3269b17c7e61: unexpected status code 404 Not Found (HEAD responses have no body, use GET for details)

      ...and more
  reason: Processing
  status: "False"
  type: RegistryContainsRequiredImages
```

8. Upload the images to the local registry using the `d8 mirror` command. Image upload to the local registry is performed via Ingress at `registry.${PUBLIC_DOMAIN}`.

   Get the password for the read-write user of the local registry:

```
$ d8 k -n d8-system get secret/registry-user-rw -o json | jq -r '.data | to_entries[] | "\(.key): \(.value | @base64d)"'
name: rw
password: KFVxXZGuqKkkumPz
passwordHash: $2a$10$Phjbr6iinLf00ZZDD2Y7O.p9H3nDOgYzFmpYKW5eydGvIsdaHQY0a
```

   Upload images to the local registry:

```
d8 mirror push \
--registry-login="rw" \
--registry-password="KFVxXZGuqKkkumPz" \
/home/user/d8-bundle \
registry.${PUBLIC_DOMAIN}/system/deckhouse
```

9. Check the registry switch status in the `registry-state` secret using this guide. After uploading the images, the `RegistryContainsRequiredImages` status should be in the `Ready` state.

Example output:

```
conditions:
# ...
- lastTransitionTime: "..."
  message: |-
    Mode: Default
    master-1: all 166 items are checked
  reason: Ready
  status: "True"
  type: RegistryContainsRequiredImages
hash: ..
mode: Direct
target_mode: Local
```

10. Wait for the switch to complete. To check the switch status, use this guide.

    Example output:

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

Switching to the `Unmanaged` Mode

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

✖ Changing the registry in `Unmanaged` mode will result in a full restart of all DKP components.

1. Before switching, perform the migration to use the `registry` module.
2. Make sure the `registry` module is enabled and running. To do this, execute the following command:

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

   Example output:

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

3. Ensure the Deckhouse job queue is empty and contains no errors:

```
d8 system queue list
```

   Example output:

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

4. Set the `Unmanaged` mode configuration in the ModuleConfig `deckhouse`. If you're using a registry other than `registry.deckhouse.io`, refer to the `deckhouse` module documentation for correct configuration.

   Configuration example:

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

5. Check the registry switch status in the `registry-state` secret using this guide.

   Example output:

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

6. If you need to switch back to the old registry management method, refer to the instruction.

⚠ This is a deprecated format for registry management.

The registry-packages-proxy module

i The module lifecycle stage: General Availability

The `registry-packages-proxy` module provides an in-cluster HTTP proxy service for accessing packages from container registries. It acts as an intermediary between cluster components and external or internal registries, offering caching capabilities to optimize bandwidth usage and improve package retrieval performance.

This module is a critical infrastructure component that runs on master nodes and is used during cluster bootstrap and runtime operations to fetch packages from container registries.

The module deploys a highly-available proxy service that:

- Runs as a deployment on master nodes with `hostNetwork` enabled to ensure availability during bootstrap when CNI is not yet available.
- Listens on port `4219` (HTTPS) on each master node's IP address.
- Provides a `/package` endpoint for retrieving registry packages by digest.
- Implements local caching of retrieved packages (up to 1 GB) to reduce network traffic and improve performance.
- Watches ModuleSource custom resources to obtain registry credentials for different repositories.
- Uses `kube-rbac-proxy` to secure access to the proxy and metrics endpoints.
- Exposes a public HTTPS API (via Ingress) for Deckhouse CLI binaries.
- Exposes an in-cluster HTTPS API for package icons (no public Ingress).

Architecture

The proxy service consists of two containers:

1. **registry-packages-proxy**: The main proxy application that:
   - Fetches packages from remote registries using digests.
   - Caches packages locally in an ephemeral volume (1 GB max).
   - Supports authentication to registries via credentials from ModuleSource resources.
   - Provides health checks and Prometheus metrics.
   - Listens on `127.0.0.1:5080` (HTTP, internal).
2. **kube-rbac-proxy**: Provides RBAC-based access control:
   - Exposes the service on port `4219` (HTTPS).
   - Secures `/metrics` endpoint with Kubernetes RBAC authorization.
   - Secures `/package` endpoint requiring appropriate permissions.
   - Secures `/v1/images/*` (Deckhouse CLI downloads) with Kubernetes RBAC authorization.
   - Allows unauthenticated access to `/healthz`.

HTTP API

After the cluster is bootstrapped and the DNS name template is configured in the publicDomainTemplate parameter, the module creates an Ingress for `registry-packages-proxy` using that template (for example, `registry-packages-proxy.company.my` for `publicDomainTemplate: "%s.company.my"`).

The endpoints listed below are reachable in two ways:

- **Public** (via the Ingress on the `registry-packages-proxy` public host): only `/v1/images/*`.
- **In-cluster only** (via the `registry-packages-proxy.d8-cloud-instance-manager.svc` Service on port `443`, or `:4219` on each master node for bootstrap): all routes, including `/v1/packages/*`.

Package icons (`/v1/packages/`) — in-cluster only

Package icons are served **without authentication** (`kube-rbac-proxy` excludes these paths from RBAC) but are **only reachable from inside the cluster**. They are intentionally **not** routed through the public Ingress, so the public domain `registry-packages-proxy.<publicDomain>` will not serve them.

Method	Path	Description
`GET`, `HEAD`	`/v1/packages/<PACKAGE-REPOSITORY>/<PACKAGE-NAME>/metadata/icon/`	Icon of the latest semver tag
`GET`, `HEAD`	`/v1/packages/<PACKAGE-REPOSITORY>/<PACKAGE-NAME>/metadata/icon`	Same as above
`GET`, `HEAD`	`/v1/packages/<PACKAGE-REPOSITORY>/<PACKAGE-NAME>/metadata/icon/<VERSION>`	Icon of a specific version (`<VERSION>` is a semantic version, e.g. `v1.0.1`)

`<PACKAGE-REPOSITORY>` is the `metadata.name` of a PackageRepository custom resource; its `spec.registry.repo` field tells the proxy which registry path to pull the icon from. The proxy reads the icon from the OCI image `<spec.registry.repo>/<PACKAGE-NAME>:<TAG>`.

The proxy looks for the following files inside the package image, in this priority order, and returns the first one it finds:

Path	Content-Type
`docs/icon.svg`	`image/svg+xml`
`docs/icon.png`	`image/png`
`docs/icon.jpg`	`image/jpeg`
`docs/icon.jpeg`	`image/jpeg`

If none of these are present (or the file is larger than 4 MiB), the proxy returns `404 Not Found`; callers should fall back to a default icon. SVG is preferred because it is resolution-independent.

Example request from a pod in the cluster:

```
curl -fsSk "https://registry-packages-proxy.d8-cloud-instance-manager.svc/v1/packages/my-repo/my-module/metadata/icon/"
```

Example response headers (when the image ships `docs/icon.svg`):

```
Content-Type: image/svg+xml
Content-Disposition: attachment; filename="<PACKAGE-NAME>.svg"
```

Deckhouse CLI downloads (`/v1/images/`) — public

These endpoints are reachable through the public Ingress (`registry-packages-proxy.<publicDomain>`) and require a valid Kubernetes token (or client certificate accepted by `kube-rbac-proxy`) and RBAC permission to `get` the `deployments/cli-binary` subresource `registry-packages-proxy` in namespace `d8-cloud-instance-manager`.

Grant access with the ClusterRole `d8:registry-packages-proxy:cli-download` (bind it to users or ServiceAccounts via ClusterRoleBinding or RoleBinding).

Method	Path	Description
`GET`	`/v1/images/<IMAGE>/tags`	JSON list of tags
`GET`, `HEAD`	`/v1/images/<IMAGE>/tags/<TAG>`	OCI image as `application/x-gzip` (flattened layers)

Allowed `<IMAGE>` values:

- `deckhouse-cli`
- `deckhouse-cli/plugins/<PLUGIN>` (single path segment for `<PLUGIN>`)

Example:

```
curl -fsS -H "Authorization: Bearer ${TOKEN}" \
  "https://registry-packages-proxy.example.com/v1/images/deckhouse-cli/tags"
```

Internal `/package` endpoint

The legacy `/package?digest=...` endpoint (used during bootstrap and by internal components) remains protected by RBAC (`deployments/http` subresource). It is not exposed through the public Ingress.

Package retrieval flow

When a component requests a package:

1. The request includes a `digest` parameter (required) and optional `repository` and `path` parameters.
2. The proxy checks its local cache for the requested digest.
3. If cached, the package is served directly from cache.
4. If not cached:
   - The proxy fetches credentials for the specified registry from watched ModuleSource resources.
   - The package is retrieved from the remote registry.
   - The package is streamed to the client while simultaneously being cached for future requests.
5. Responses include appropriate HTTP headers for caching (`Cache-Control`, `ETag`, `Content-Length`).

High availability

The module ensures high availability through:

- Running multiple replicas on master nodes (in HA configurations).
- Pod anti-affinity rules to distribute pods across different masters.
- PodDisruptionBudget to prevent simultaneous disruption of all replicas.
- Vertical Pod Autoscaler support for automatic resource adjustments.

RBAC roles provided by the module

ClusterRole	Purpose
`d8:registry-packages-proxy:cli-download`	Access to `/v1/images/*`
`d8:registry-packages-proxy:packages-download`	Reserved for future authenticated `/v1/packages/*` routes (icons are served anonymously, in-cluster only)

Limitations

- The module runs exclusively on master nodes.
- It requires `hostNetwork: true` to function during bootstrap phase.
- Cache size is limited to 1 GB per pod.
- Most HTTP endpoints require Kubernetes RBAC; only health checks (healthz) and package icons are anonymous. Package icons are additionally limited to in-cluster access (no public Ingress route).
- Package icons are read from fixed paths inside the package image (`docs/icon.svg`, `docs/icon.png`, `docs/icon.jpg`, `docs/icon.jpeg`) with SVG preferred. Maximum icon size is 4 MiB.

The registry-packages-proxy: configuration

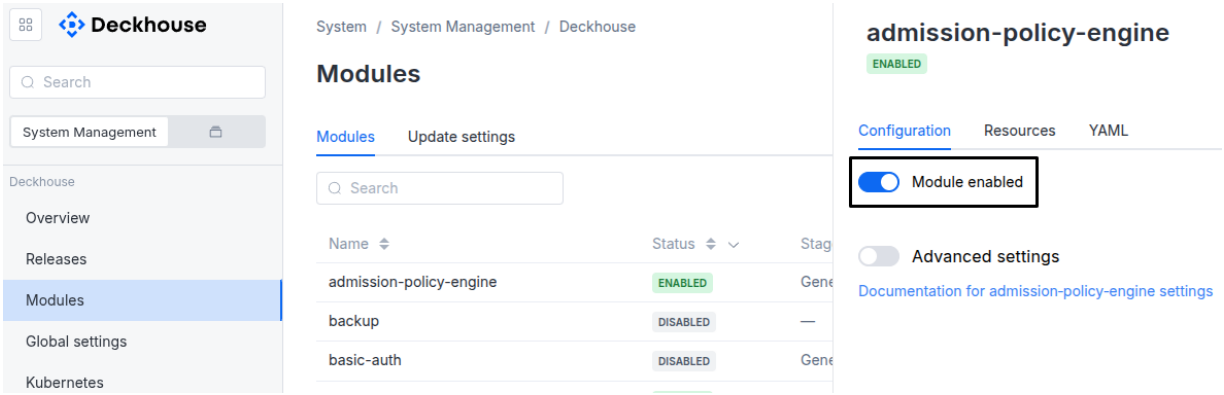
> The module lifecycle stage: General Availability

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `registry-packages-proxy` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable registry-packages-proxy
```

- **Using ModuleConfig `registry-packages-proxy`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `registry-packages-proxy` (create it if necessary);

  Example of a manifest to enable module `registry-packages-proxy`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry-packages-proxy
spec:
  enabled: true
```

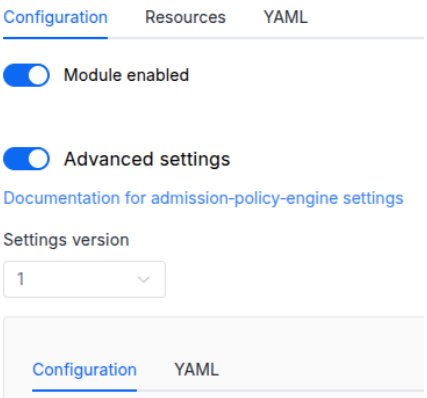
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `registry-packages-proxy` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



  You can also edit the ModuleConfig object `registry-packages-proxy` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `registry-packages-proxy`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `registry-packages-proxy` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

```
d8 k edit mc registry-packages-proxy
```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

You can also create a file with manifest for ModuleConfig `registry-packages-proxy` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

Apply the manifest using the following command (indicate the manifest file name):

```
d8 k apply -f <FILENAME>
```

Example of a manifest for ModuleConfig `registry-packages-proxy`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: registry-packages-proxy
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

The module does not have any settings.

The secret-copier module

> **i** The module lifecycle stage: General Availability

This module copies Secrets to all namespaces.

It facilitates the copying of Secrets required to pull images and to provision Ceph's RBDs as part of the CI process.

> **⚠** The `secret-copier` module cannot be used together with `multitenancy-manager`.
>
> `multitenancy-manager` creates isolated environments for users within their projects, while `secret-copier` automatically distributes secrets across all namespaces. This may lead to sensitive data leaks if important secrets end up in a user's isolated environment, violating the security model.
>
> If you need to provide a shared certificate (e.g., a WC certificate for an internal environment) or a shared registry access token, do not use `secret-copier`. Instead, place such secrets in the project template in `multitenancy-manager` — the cluster administrator should define them in the project configuration.

How does it work?

This module monitors the Secrets (with the `secret-copier.deckhouse.io/enabled: ""` label) in the `default` namespace for changes.

- The module copies such a Secret to all namespaces after it is created.
- When a Secret is changed, its new contents are also propagated to all namespaces.
- When a Secret is deleted, its copies are deleted from all namespaces.
- The module restores the original content of a copy of a Secret in the application namespace if it gets modified.
- The module copies all the Secrets (that have the `secret-copier.deckhouse.io/enabled: ""` label) of the `default` namespace to any newly created namespace.

Additionally, it synchronizes the Secrets every night, ensuring they are identical to those in the `default` namespace.

What do I need to configure?

All you need to do is to create a Secret with the `secret-copier.deckhouse.io/enabled: ""` label in the `default` namespace.

> **i** **Note!** The working namespace for the module is `default`, Secrets will be copied only from it. Secrets with the secret-copier.deckhouse.io/enabled: "" label created in other namespaces will be automatically deleted when the module is enabled.

How to synchronize Secret to some selected namespaces instead of all namespaces?

Specify namespace label-selector in the value of the `secret-copier.deckhouse.io/target-namespace-selector` annotation. For example: `secret-copier.deckhouse.io/target-namespace-selector: "app=custom"`. The module will create a copy of that Secret in all namespaces that matches the label-selector.

The service-with-healthchecks module

> **i** The module lifecycle stage: General Availability

Limitations of the built-in Service load balancer

In Kubernetes, the `Service` resource handles the internal and external load balancing of requests. It routes requests between the application's worker pods and excludes failed instances from load balancing. The readiness probes defined in the specs of the containers running in this pod make sure that the pod can handle incoming traffic.

The built-in Service load balancer is suitable for most cloud application tasks, but it has two limitations:

- If at least one container in a pod fails the readiness probe, the entire pod is marked as `NotReady` and is excluded from load balancing.
- You can only define one probe for each container, so it is not possible to create independent probes to check, for example, whether reads and writes are available.

The following are examples of scenarios where the capabilities of the regular load balancer are inadequate:

- Database:
  - Runs like a service consisting of three pods, `db-0`, `db-1`, and `db-2`. Each pod contains one container with a running database process.
  - You would like to create two Services, `db-write` for writing and `db-read` for reading.
  - Read queries must be load balanced across all pods.
  - Write queries are only routed to the database's master pod.

- Virtual machine:
  - The pod contains a single container running the `qemu` process, which acts as a hypervisor for the guest virtual machine.
  - The guest virtual machine is running some independent processes, such as a web server and an SMTP server.
  - You would like to create two Services, `web` and `smtp`, and define a separate readiness probe for each service.

ServiceWithHealthcheck load balancer capabilities

Unlike the regular load balancer, in which readiness probes are tied to the container state, `ServiceWithHealthcheck` allows you to set up active probes on individual TCP ports. In this way, each of the load balancers dealing with the same pod can operate independently of the others.

You can configure this balancing method using the ServiceWithHealthchecks resource:

- Its specification is the same as the regular `Service` except for the `healthcheck` section, which contains a set of probes.
- Currently, three types of probes are supported:
  - `TCP` — a regular probe that establishes a TCP connection.
  - `HTTP` — a probe that sends an HTTP request and waits for a specific response code.
  - `PostgreSQL` — a probe that sends an SQL query and waits for it to complete successfully.

Examples can be found in the documentation.

How the ServiceWithHealthcheck load balancer works

The load balancer is made up of two components:

- The **controller** runs on the cluster master nodes and manages `ServiceWithHealthcheck` resources,
- The agents operate on each cluster node and carry out probing for pods that run on that node.

The ServiceWithHealthcheck load balancer is designed to be CNI implementation agnostic. It uses the built-in K8s `Service` and `EndpointSlice` resources:

- When creating a `ServiceWithHealthcheck` resource, the controller automatically creates an eponymous Service resource in the same namespace with an empty `selector` field. This prevents the default controller from creating `EndpointSlice`, which are used to configure load balancing.
- When pods subject to `ServiceWithHealthcheck` are scheduled to a particular node, the agent running on that node runs the pre-configured probes and creates an `EndpointSlice` for them with a list of IP addresses and ports to be checked. This `EndpointSlice` is bound to the `Service` child resource created above.
- CNI maps all `EndpointSlice` to the regular services created above and performs load balancing across probed IP addresses and ports on all nodes in the cluster.

Migrating from a Service to a ServiceWithHealthchecks resource, for example within the framework of CI/CD, should not cause difficulties. The ServiceWithHealthchecks specification basically repeats the Service specification, but contains an additional healthchecks section. During the lifecycle of the ServiceWithHealthchecks resource, a service of the same name is created in the same namespace in order to direct traffic to workloads in the cluster in the usual way (kube-proxy or cni).

Configuration

i The module lifecycle stage: General Availability

i For the ServiceWithHealthchecks load balancers you create to work, the following conditions must be met:

- The network policy of the custom project in which the ServiceWithHealthchecks will be created must include a rule allowing incoming traffic from all pods in the `d8-service-with-healthchecks` namespace:

  ```
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: d8-service-with-healthchecks
  ```

  For more information on network policies, see the Configuring Network Policies section.

- The cluster role used in ClusterRoleBinding and RoleBinding when assigning permissions to users and service accounts for the ServiceWithHealthchecks resource must be extended with the following rules:

  - `get`
  - `list`
  - `watch`
  - `create`
  - `update`
  - `patch`
  - `delete`.

  For more details, see the section Granting permissions to users and service accounts.

⚠ Enabling the module does not automatically replace existing Service resources with ServiceWithHealthcheck resources. To replace existing services with ServiceWithHealthcheck, follow these steps:

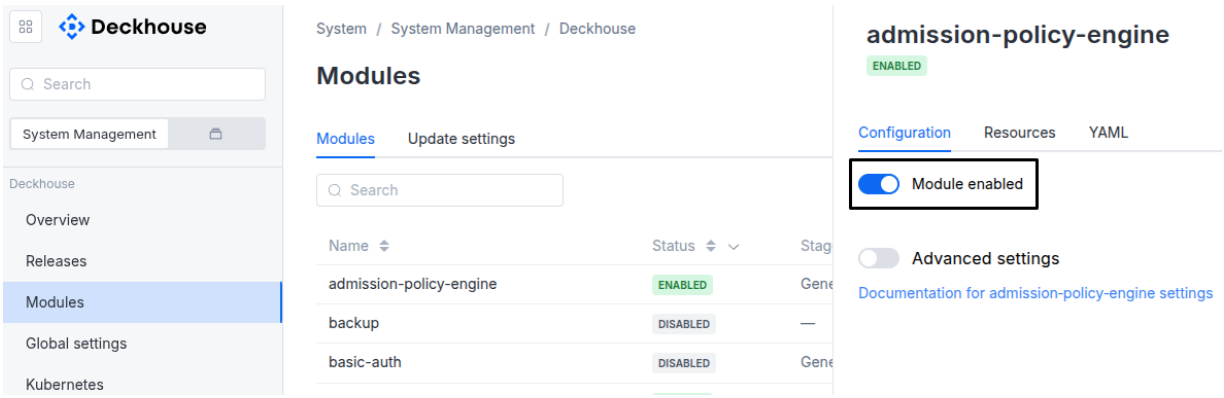
- Create ServiceWithHealthcheck resources with the same names and parameters as the existing Service resources you want to replace. When creating a ServiceWithHealthcheck, specify the required `healthchecks` parameters.
- Delete the Service resources that you want to replace with ServiceWithHealthcheck.

The module is not enabled by default in any bundles.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `service-with-healthchecks` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable service-with-healthchecks
  ```

- **Using ModuleConfig `service-with-healthchecks`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `service-with-healthchecks` (create it if necessary);

  Example of a manifest to enable module `service-with-healthchecks`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: service-with-healthchecks
  spec:
    enabled: true
  ```

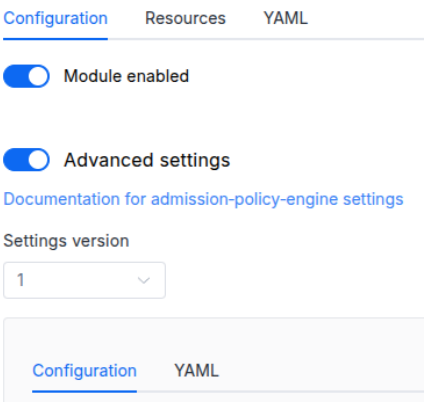
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `service-with-healthchecks` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `service-with-healthchecks` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `service-with-healthchecks`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `service-with-healthchecks` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc service-with-healthchecks
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `service-with-healthchecks` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `service-with-healthchecks`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: service-with-healthchecks
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.debug**
    boolean

    Option to instruct the logger to output stacktraces for warnings and errors.

    Default: `false`

The service-with-healthchecks: Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

ServiceWithHealthchecks

Short names: swh

Scope: Namespaced
Version: v1alpha1

ServiceWithHealthchecks is the schema for the `servicewithhealthchecks` API.

- **apiVersion**
  string
- **kind**
  string
- **metadata**
  object
- **spec**
  object

  Defines the desired state of ServiceWithHealthchecks.

  For the ServiceWithHealthchecks load balancers you create to work, the following conditions must be met:

  - The network policy of the custom project in which the ServiceWithHealthchecks will be created must include a rule allowing incoming traffic from all pods in the `d8-service-with-healthchecks` namespace:

    ```
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: d8-service-with-healthchecks
    ```

    For more information on network policies, see the Configuring Network Policies section.

  - The cluster role used in ClusterRoleBinding and RoleBinding when assigning permissions to users and service accounts for the ServiceWithHealthchecks resource must be extended with the following rules:

    - `get`
    - `list`
    - `watch`
    - `create`
    - `update`
    - `patch`
    - `delete`.

    For more details, see the section Granting permissions to users and service accounts.

  - **spec.allocateLoadBalancerNodePorts**
    boolean

    Determines whether NodePort should be automatically allocated for services of LoadBalancer type. Default is `true`. It may be set to `false` if the cluster load-balancer does not rely on NodePorts.

    If the caller requests specific NodePorts (by specifying a value), those requests will be respected, regardless of this field. This parameter can be set only for services of LoadBalancer type and is automatically reset when the type is changed to another.

  - **spec.clusterIP**
    string

    The IP address of the service.

    Normally, the IP address is randomly assigned to the service. If the IP address is manually specified, it must be within the valid range (specified by the system configuration) and free for assignment; otherwise, service creation will fail.

    This field may not be changed through updates unless the type field is also being changed to `ExternalName` (which requires this field to be blank) or the type field is being changed from `ExternalName` (in which case this field may optionally be specified, as describe above).

Valid values are `None`, empty string (“”), or a valid IP address. Setting this to `None` makes a “headless service” (no virtual IP), which is useful when direct endpoint connections are preferred and proxying is not required. Only applies to types ClusterIP, NodePort, and LoadBalancer. If this field is specified when creating a Service of type ExternalName, creation will fail. This field will be wiped when updating a Service to type ExternalName. More info.

- **spec.externalIPs**
  array of strings

  A list of IP addresses where nodes in the cluster will also accept traffic for this service. These IPs are not managed by Kubernetes. The user is responsible for ensuring that traffic arrives at a node with this IP.

  A common example is external load-balancers that are not part of the Kubernetes system.

- **spec.externalName**
  string

  An external reference that discovery mechanisms will return as an alias for this service (e.g., a DNS record CNAME).

  Must be a lowercase RFC-1123 hostname and requires `type` to be “ExternalName”.

- **spec.externalTrafficPolicy**
  string

  Describes how nodes distribute service traffic they receive on one of the Service’s “externally-facing” addresses (NodePorts, ExternalIPs, and LoadBalancer IPs). If set to `Local`, the proxy will configure the service in a way that assumes that external load balancers will take care of balancing the service traffic between nodes, and so each node will deliver traffic only to the node-local endpoints of the service, without masquerading the client source IP. Traffic mistakenly sent to a node with no endpoints will be dropped.

  The default value, `Cluster`, uses the standard behavior of routing to all endpoints evenly (possibly modified by topology and other features).

  Note that traffic sent to an External IP or LoadBalancer IP from within the cluster will always get `Cluster` semantics, but clients sending to a NodePort from within the cluster may need to take traffic policy into account when picking a node.

- **spec.healthCheckNodePort**
  integer

  Specifies the healthcheck nodePort for the service. This only applies when type is set to LoadBalancer and `externalTrafficPolicy` is set to `Local`. If a value is specified, is in-range, and is not in use, it will be used.

  If not specified, a value will be automatically allocated. External systems (e.g. load-balancers) can use this port to determine if a given node holds endpoints for this service or not.

  If this field is specified when creating a Service which does not need it, creation will fail. This field will be wiped when updating a Service to no longer need it (e.g. changing type).

  This field cannot be updated once set.

- **spec.healthcheck**
  object

  Required value

  - **spec.healthcheck.initialDelaySeconds**
    integer

    Number of seconds after the container has started before liveness probes are initiated.

    Default: `10`

    Allowed values: `0 <= X <= 108000`

  - **spec.healthcheck.periodSeconds**
    integer

    How often (in seconds) to perform the probe.

    Default: `10`

    Allowed values: `1 <= X <= 108000`

  - **spec.healthcheck.probes**
    array of objects
    - **spec.healthcheck.probes.failureThreshold**
      integer

      Minimum consecutive failures for the probe to be considered failed after having succeeded.

      Default: `3`

      Allowed values: `1 <= X`

    - **spec.healthcheck.probes.http**
      object
      - **spec.healthcheck.probes.http.caCert**
        string

        CA certificate for HTTPS connection.

      - **spec.healthcheck.probes.http.host**
        string

        Hostname to connect to, defaults to the pod’s IP address.

      - **spec.healthcheck.probes.http.httpHeaders**
        array of objects

        Custom headers that will be added to the HTTP request and used when performing the probe. Duplicate headers are allowed.

        - **spec.healthcheck.probes.http.httpHeaders.name**
          string

          Required value

          The header name.

          This will be canonicalized upon output, so case-variant names will be understood as the same header.

        - **spec.healthcheck.probes.http.httpHeaders.value**
          string

          Required value

          The header value.
      - **spec.healthcheck.probes.http.insecureSkipTLSVerify**
        boolean

        Indicates if the client should skip verifying the server's TLS certificate chain and hostname.

        Default: `false`
      - **spec.healthcheck.probes.http.method**
        string

        Method to use for the request.

        Default: `GET`

        Allowed values: `GET`, `HEAD`
      - **spec.healthcheck.probes.http.path**
        string

        Path to access on the HTTP server.
      - **spec.healthcheck.probes.http.scheme**
        string

        Scheme to use for connecting to the host.

        Default: `HTTP`

        Allowed values: `HTTP`, `HTTPS`
      - **spec.healthcheck.probes.http.targetPort**
        integer

        Required value

        Number of the port to access on the container.

        Allowed values: `1 <= X <= 65535`
  - **spec.healthcheck.probes.mode**
    string

    The mode for the probe.

    Allowed values: `HTTP`, `TCP`, `PostgreSQL`
  - **spec.healthcheck.probes.postgreSQL**
    object
    - **spec.healthcheck.probes.postgreSQL.authSecretName**
      string

      The name of the secret located in the same namespace as this resource.

      The secret can contain the following authentication fields: `tlsMode`, `clientCert`, `clientKey`, `caCert`, `password`, `user`.

      > **Warining.** The type of the secret must be `network.deckhouse.io/postgresql-credentials`.
    - **spec.healthcheck.probes.postgreSQL.dbName**
      string

      Optional Database Name.
    - **spec.healthcheck.probes.postgreSQL.query**
      string

      Query to PostgreSQL for testing.

      Must return one of the values `TRUE`: `t`, `true`, `y`, `yes`, `1`. to be considered successful.

      > **Warining.** The privileges for the user under whom the requests are executed should be minimal.

      Default: `select 1`
    - **spec.healthcheck.probes.postgreSQL.targetPort**
      integer

      Required value

      Number of the port to access on the container.

      Allowed values: `1 <= X <= 65535`
  - **spec.healthcheck.probes.successThreshold**
    integer

    Minimum consecutive successes for the probe to be considered successful after having failed.

    Default: `1`

    Allowed values: `1 <= X`
  - **spec.healthcheck.probes.tcp**
    object
    - **spec.healthcheck.probes.tcp.targetPort**
      integer

      Required value

Number of the port to access on the container.

Allowed values: `1 <= X <= 65535`

- **spec.healthcheck.probes.timeoutSeconds**
  integer

  Number of seconds after which the probe times out.

  Default: `1`

  Allowed values: `1 <= X`

- **spec.internalTrafficPolicy**
  string

  Describes how nodes distribute service traffic they receive on the ClusterIP.

  If set to `Local`, the proxy will assume that pods only want to talk to endpoints of the service on the same node as the pod, dropping the traffic if there are no local endpoints.

  The default value, `Cluster`, uses the standard behavior of routing to all endpoints evenly (possibly modified by topology and other features).

- **spec.ipFamilies**
  array of strings

  A family of IP addresses (e.g., IPv4, IPv6) assigned to a service. This field is usually assigned automatically based on cluster configuration and the `ipFamilyPolicy` field.

  If this field is specified manually, the requested family is available in the cluster, and `ipFamilyPolicy` allows it, it will be used; otherwise creation of the service will fail. This field is conditionally mutable: it allows for adding or removing a secondary IP family, but it does not allow changing the primary IP family of the Service.

  Valid values are `IPv4` and `IPv6`.This field only applies to Services of types ClusterIP, NodePort, and LoadBalancer, and does apply to “headless” services.

  This field will be wiped when updating a Service to type `ExternalName`.

  This field may hold a maximum of two entries (dual-stack families, in either order). These families must correspond to the values of the `clusterIPs` field, if specified. Both `clusterIPs` and `ipFamilies` are governed by the `ipFamilyPolicy` field.

  - **Element of the array**
    string

    Represents the IP Family (IPv4 or IPv6).

    This type is used to express the family of an IP expressed by a type (e.g. service.spec.ipFamilies).

- **spec.ipFamilyPolicy**
  string

  Required or requested dual stack support for the service.

  If there is no value provided, then this field will be set to `SingleStack`. Services can be `SingleStack` (a single IP family), `PreferDualStack` (two IP families on dual-stack configured clusters or a single IP family on single-stack clusters), or `RequireDualStack` (two IP families on dual-stack configured clusters, otherwise fail).

  The `ipFamilies` and `clusterIPs` fields depend on the value of this field. This field will be wiped when updating a service to type `ExternalName`.

- **spec.loadBalancerClass**
  string

  The class of the load balancer implementation this Service belongs to. If specified, the value of this field must be a label-style identifier, with an optional prefix, e.g. `internal-vip` or `example.com/internal-vip`. Unprefixed names are reserved for end-users.

  This field can only be set when the Service type is `LoadBalancer`.

  If not set, the default load balancer implementation is used, today this is typically done through the cloud provider integration, but should apply for any default implementation.

  If set, it is assumed that a load balancer implementation is watching for Services with a matching class. Any default load balancer implementation (e.g. cloud providers) should ignore Services that set this field. This field can only be set when creating or updating a Service to type `LoadBalancer`.

  Once set, it can not be changed. This field will be wiped when a service is updated to a non `LoadBalancer` type.

- **spec.loadBalancerIP**
  string

  Only applies to Service Type: LoadBalancer. This feature depends on whether the underlying cloud-provider supports specifying the loadBalancerIP when a load balancer is created.

  This field will be ignored if the cloud-provider does not support the feature.

- **spec.loadBalancerSourceRanges**
  array of strings

  If specified and supported by the platform, this will restrict traffic through the cloud-provider load-balancer will be restricted to the specified client IPs.

  This field will be ignored if the cloud-provider does not support the feature. More info.

- **spec.ports**
  array of objects

  The list of ports that are exposed by this service. More info.

  ServicePort contains information on service’s port.

  - **spec.ports.appProtocol**
    string

    The application protocol for this port. This is used as a hint for implementations to offer richer behavior for protocols that they understand. This field follows standard Kubernetes label syntax. Valid values are either:

    - Un-prefixed protocol names - reserved for IANA standard service names as RFC-6335.
    - Kubernetes-defined prefixed names:

        - `kubernetes.io/h2c` — HTTP/2 prior knowledge over cleartext as described in RFC;
        - `kubernetes.io/ws` — WebSocket over cleartext as described in RFC;
        - `kubernetes.io/wss` — WebSocket over TLS as described in RFC.
      - Other protocols should use implementation-defined prefixed names such as `mycompany.com/my-custom-protocol`.
    - **spec.ports.name**
      string

      The name of this port within the service. This must be a `DNS_LABEL`.

      All ports within a ServiceSpec must have unique names. When considering the endpoints for a Service, this must match the 'name' field in the `EndpointPort`. Optional if only one `ServicePort` is defined on this service.

    - **spec.ports.nodePort**
      integer

      The port on each node on which this service is exposed when type is NodePort or LoadBalancer. Usually assigned by the system. If a value is specified, in-range, and not in use it will be used, otherwise the operation will fail.

      If not specified, a port will be allocated if this Service requires one.

      If this field is specified when creating a Service which does not need it, creation will fail.

      This field will be wiped when updating a Service to no longer need it (e.g. changing type from NodePort to ClusterIP). More info.

    - **spec.ports.port**
      integer

      Required value

      The port that will be exposed by this service.

    - **spec.ports.protocol**
      string

      The IP protocol for this port. Supports `TCP`, `UDP`, and `SCTP`.

      Default: `TCP`

    - **spec.ports.targetPort**
      integer

      Number of the port to access on the pods targeted by the service.

      If this is not specified, the value of the `port` field is used (an identity map).

      This field is ignored for services with `clusterIP=None`, and should be omitted or set equal to the `port` field. More info.

- **spec.publishNotReadyAddresses**
  boolean

  Indicates that any agent which deals with endpoints for this Service should disregard any indications of ready/not-ready. The primary use case for setting this field is for a StatefulSet's Headless Service to propagate SRV DNS records for its Pods for the purpose of peer discovery.

  The Kubernetes controllers that generate `Endpoints` and `EndpointSlice` resources for Services interpret this to mean that all endpoints are considered "ready" even if the Pods themselves are not. Agents which consume only Kubernetes generated endpoints through the `Endpoints` or `EndpointSlice` resources can safely assume this behavior.

- **spec.selector**
  object

  Route service traffic to pods with label keys and values matching this selector. If empty or not present, the service is assumed to have an external process managing its endpoints, which Kubernetes will not modify.

  Only applies to types ClusterIP, NodePort, and LoadBalancer.

  Ignored if type is `ExternalName`. More info.

- **spec.sessionAffinity**
  string

  Used to maintain session affinity. Enable client IP based session affinity. Must be `ClientIP` or `None`. More info.

- **spec.sessionAffinityConfig**
  object

  Contains the configurations of session affinity.

  - **spec.sessionAffinityConfig.clientIP**
    object

    Contains the configurations of `ClientIP` based session affinity.

    - **spec.sessionAffinityConfig.clientIP.timeoutSeconds**
      integer

      Specifies the seconds of `ClientIP` type session sticky time. The value must be `>0 && <=86400` (for 1 day) if `ServiceAffinity == ClientIP`.

- **spec.trafficDistribution**
  string

  Offers a way to express preferences for how traffic is distributed to Service endpoints. Implementations can use this field as a hint, but are not required to guarantee strict adherence.

  If the field is not set, the implementation will apply its default routing strategy. If set to `PreferClose`, implementations should prioritize endpoints that are topologically close (e.g., same zone).

- **spec.type**
  string

  Determines how the Service is exposed.

  Defaults to ClusterIP. Valid options are ExternalName, ClusterIP, NodePort, and LoadBalancer.

  ClusterIP allocates a cluster-internal IP address for load-balancing to endpoints. Endpoints are determined by the `selector` or if that is not specified, by manual construction of an `Endpoints` object or `EndpointSlice` objects.

  If `clusterIP` is `None`, no virtual IP is allocated and the endpoints are published as a set of endpoints rather than a virtual IP.

NodePort builds on ClusterIP and allocates a port on every node which routes to the same endpoints as the `clusterIP`.

LoadBalancer builds on NodePort and creates an external load-balancer (if supported in the current cloud) which routes to the same endpoints as the `clusterIP`.

ExternalName aliases this service to the specified `externalName`. Several other fields do not apply to ExternalName services. More info.

- **status**
  object

  Defines the observed state of ServiceWithHealthchecks

  - **status.conditions**
    array of objects

    Current service state

    Condition contains details for one aspect of the current state of this API Resource. This struct is intended for direct use as an array at the field path `.status.conditions`.

    - **status.conditions.lastTransitionTime**
      string

      Required value

      Is the last time the condition transitioned from one status to another. This should be when the underlying condition changed.

      If that is not known, then using the time when the API field changed is acceptable.

    - **status.conditions.message**
      string

      Required value

      Is a human readable message indicating details about the transition.

      This may be an empty string.

      Maximum length: `32768`

    - **status.conditions.observedGeneration**
      integer

      Represents the .metadata.generation that the condition was set based upon. For instance, if .metadata.generation is currently 12, but the .status.conditions[x].observedGeneration is 9, the condition is out of date with respect to the current state of the instance.

      Allowed values: `0 <= X`

    - **status.conditions.reason**
      string

      Required value

      Contains a programmatic identifier indicating the reason for the condition's last transition. Producers of specific condition types may define expected values and meanings for this field, and whether the values are considered a guaranteed API.

      The value should be a CamelCase string.

      This field may not be empty.

      Pattern: `^[A-Za-z]([A-Za-z0-9_,:]*[A-Za-z0-9_])?$`

      Length: `1..1024`

    - **status.conditions.status**
      string

      Required value

      status of the condition, one of `True`, `False`, `Unknown`.

      Allowed values: `True`, `False`, `Unknown`

    - **status.conditions.type**
      string

      Required value

      Condition in CamelCase or in `foo.example.com/CamelCase`.

      Many `.condition.type` values are consistent across resources like Available, but because arbitrary conditions can be useful (see .node.status.conditions), the ability to deconflict is important. The regex it matches is `(dns1123SubdomainFmt/)?(qualifiedNameFmt)`.

      Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

      Maximum length: `316`

  - **status.endpointStatuses**
    array of objects

    Current service state

    - **status.endpointStatuses.failedProbes**
      array of strings
    - **status.endpointStatuses.lastProbeTime**
      string
    - **status.endpointStatuses.nodeName**
      string
    - **status.endpointStatuses.podName**
      string

      Required value

    - **status.endpointStatuses.probesSuccessful**
      boolean

        - **status.endpointStatuses.ready**
          boolean
    - **status.healthcheckCondition**
      object
        - **status.healthcheckCondition.activeNodeName**
          string
        - **status.healthcheckCondition.endpoints**
          integer
        - **status.healthcheckCondition.observedGeneration**
          integer
        - **status.healthcheckCondition.readyEndpoints**
          integer
    - **status.loadBalancer**
      object

      Contains the current status of the load-balancer, if one is present.

        - **status.loadBalancer.ingress**
          array of objects

          Is a list containing ingress points for the load-balancer. Traffic intended for the service should be sent to these ingress points.

          LoadBalancerIngress represents the status of a load-balancer ingress point: traffic intended for the service should be sent to an ingress point.

            - **status.loadBalancer.ingress.hostname**
              string

              Is set for load-balancer ingress points that are DNS based (typically AWS load-balancers)

            - **status.loadBalancer.ingress.ip**
              string

              Is set for load-balancer ingress points that are IP based (typically GCE or OpenStack load-balancers)

            - **status.loadBalancer.ingress.ipMode**
              string

              Specifies how the load-balancer IP behaves, and may only be specified when the ip field is specified. Setting this to `VIP` indicates that traffic is delivered to the node with the destination set to the load-balancer's IP and port.

              Setting this to `Proxy` indicates that traffic is delivered to the node or pod with the destination set to the node's IP and node port or the pod's IP and port.

              Service implementations may use this information to adjust traffic routing.

            - **status.loadBalancer.ingress.ports**
              array of objects

              Is a list of records of service ports

              If used, every port defined in the service should have an entry in it

                - **status.loadBalancer.ingress.ports.error**
                  string

                  Error is to record the problem with the service NodePorts.

                  The format of the error shall comply with the following rules:

                    - built-in error values shall be specified in this file and those shall use CamelCase names
                    - cloud provider specific error values must have names that comply with the format `foo.example.com/CamelCase`.

                  The regex it matches is `(dns1123SubdomainFmt/)?(qualifiedNameFmt)`.

                  Pattern: `^([a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*/)?(([A-Za-z0-9][-A-Za-z0-9_.]*)?[A-Za-z0-9])$`

                  Maximum length: `316`

                - **status.loadBalancer.ingress.ports.port**
                  integer

                  Required value

                  Is the port number of the service port of which status is recorded here

                - **status.loadBalancer.ingress.ports.protocol**
                  string

                  Required value

                  Is the protocol of the service port of which status is recorded here

                  The supported values are: `TCP`, `UDP`, `SCTP`

                  Default: `TCP`

Module service-with-healthchecks: examples

> **i** The module lifecycle stage: General Availability

i For the ServiceWithHealthchecks load balancers you create to work, the following conditions must be met:

- The network policy of the custom project in which the ServiceWithHealthchecks will be created must include a rule allowing incoming traffic from all pods in the `d8-service-with-healthchecks` namespace:

  ```
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: d8-service-with-healthchecks
  ```

  For more information on network policies, see the Configuring Network Policies section.

- The cluster role used in ClusterRoleBinding and RoleBinding when assigning permissions to users and service accounts for the ServiceWithHealthchecks resource must be extended with the following rules:
  - `get`
  - `list`
  - `watch`
  - `create`
  - `update`
  - `patch`
  - `delete`.

  For more details, see the section Granting permissions to users and service accounts.

⚠ Enabling the module does not automatically replace existing Service resources with ServiceWithHealthcheck resources. To replace existing services with ServiceWithHealthcheck, follow these steps:

- Create ServiceWithHealthcheck resources with the same names and parameters as the existing Service resources you want to replace. When creating a ServiceWithHealthcheck, specify the required `healthchecks` parameters.
- Delete the Service resources that you want to replace with ServiceWithHealthcheck.

Running two independent balancers on the same virtual machine

Suppose that there are two applications running on a Linux virtual machine — an HTTP server (TCP 8080) and an SMTP server (TCP 2525). You need to set up two separate balancers for these services, a web balancer and an SMTP balancer.

Creating a virtual machine

Create a `my-vm` virtual machine by following the examples in the DVP documentation.

In the manifest example below, the `vm: my-vm` label is included so that the virtual machine can be bound to load balancers.

```
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachine
metadata:
  name: my-vm
  namespace: my-ns
  labels:
    vm: my-vm
spec:
  virtualMachineClassName: generic
  cpu:
    cores: 1
  memory:
    size: 1Gi
  provisioning:
    type: UserData
    userData: |
      #cloud-config
      ssh_pwauth: True
      users:
      - name: cloud
        passwd:
'$6$rounds=4096$saltsalt$fPmUsbjAuA7mnQNTajQM6ClhesyG0.yyQhvahas02ejfMAq1ykBo1RquzS0R6GgdIDlvS.kbUwDablGZKZcTP/'
        shell: /bin/bash
        sudo: ALL=(ALL) NOPASSWD:ALL
        lock_passwd: False
  blockDeviceRefs:
    - kind: VirtualDisk
      name: linux-disk
```

Web service and SMTP load balancer manifests

Below is an example of a manifest of a web service load balancer:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ServiceWithHealthchecks
metadata:
  name: web
  namespace: my-ns
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    vm: my-vm
  healthcheck:
    probes:
    - mode: HTTP
      http:
        targetPort: 8080
        method: GET
        path: /healthz
```

Below is an example of a manifest of a SMTP load balancer:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ServiceWithHealthchecks
metadata:
  name: smtp
  namespace: my-ns
spec:
  ports:
  - port: 25
    protocol: TCP
    targetPort: 2525
  selector:
    vm: my-vm
  healthcheck:
    probes:
    - mode: TCP
      tcp:
        targetPort: 2525
```

Load balancers for working with a PostgreSQL cluster

Creating a StatefulSet for PostgreSQL

In order for `StatefulSet` to operate properly, you will have to create a regular Service to generate the pod DNS names. This service will not be used for direct access to the database.

```
apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  selector:
    app: postgres
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
```

Below is an example of a `StatefulSet` manifest:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
  name: my-ns
spec:
  serviceName: postgres
  replicas: 3
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:13
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_USER
              value: postgres
            - name: POSTGRES_PASSWORD
              value: example
```

Configuring ServiceWithHealthchecks load balancers

Create a Secret to store credentials so that probes can access the database:

```
d8 k -n my-ns create secret generic cred-secret --from-literal=user=postgres --from-literal=password=example cred-secret
```

Below is an example of a load balancer manifest for reading:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ServiceWithHealthchecks
metadata:
  name: postgres-read
spec:
  ports:
  - port: 5432
    protocol: TCP
    targetPort: 5432
  selector:
    app: postgres
  healthcheck:
    probes:
    - mode: PostgreSQL
      postgreSQL:
        targetPort: 5432
        dbName: postgres
        authSecretName: cred-secret
        query: "SELECT 1"
```

And here is an example of a load balancer manifest for writing:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ServiceWithHealthchecks
metadata:
  name: postgres-write
spec:
  ports:
  - port: 5432
    protocol: TCP
    targetPort: 5432
  selector:
    app: postgres
  healthcheck:
    probes:
    - mode: PostgreSQL
      postgreSQL:
        targetPort: 5432
        dbName: postgres
        authSecretName: cred-secret
        query: "SELECT NOT pg_is_in_recovery()"
```

The terraform-manager module

> **i** The module lifecycle stage: General Availability

The module is responsible for monitoring and synchronizing the state of the underlying infrastructure and persistent nodes in the cloud

environment. The implementation is based on Terraform and is used in the Deckhouse Kubernetes Platform for interaction with supported cloud providers.

- The module consists of 2 parts:
  - `terraform-auto-converger` — checks the Terraform state and applies non-destructive changes;
  - `terraform-state-exporter` — checks the Terraform state and exports cluster metrics.
- The module is enabled by default if the following secrets are present in the cluster:
  - `kube-system/d8-provider-cluster-configuration`;
  - `d8-system/d8-cluster-terraform-state`.

Third-party components

List of third-party software used in the `terraform-manager` module:

- Terraform

  License: Mozilla Public License 2.0

  A tool for building, changing, and versioning infrastructure safely and efficiently.

Configuration

> **i** The module lifecycle stage: General Availability

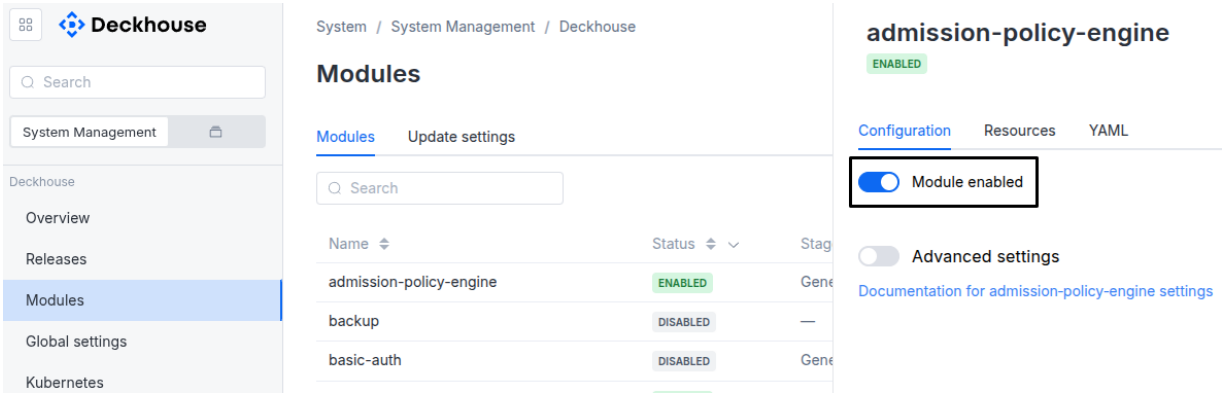
The module has 10 alerts.

The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `terraform-manager` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable terraform-manager
```

- **Using ModuleConfig `terraform-manager`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `terraform-manager` (create it if necessary);

  Example of a manifest to enable module `terraform-manager`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: terraform-manager
spec:
  enabled: true
```

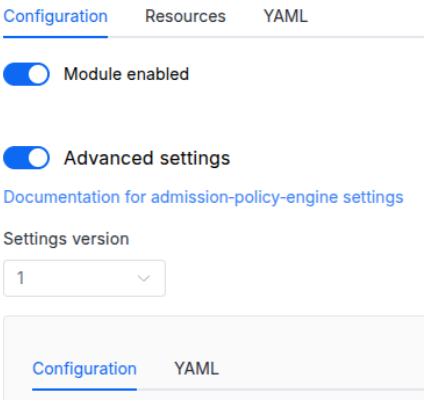
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `terraform-manager` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `terraform-manager` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `terraform-manager`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `terraform-manager` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc terraform-manager
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `terraform-manager` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `terraform-manager`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: terraform-manager
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.autoConvergerEnabled**
    boolean

    Enables periodic checking of Terraform state and applying non-destructive changes.

    The frequency of the check is set by the autoConvergerPeriod parameter.

    Default: `true`
  - **settings.autoConvergerPeriod**
    string

    The time interval after which the state of Terraform is checked and applied.

    It is specified as a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

    Default: `1h`

    Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`
  - **settings.nodeSelector**
    object

    The same as in the Pod’s `spec.nodeSelector` parameter in Kubernetes.

    If the parameter is omitted or `false`, it will be determined automatically.
  - **settings.tolerations**
    array of objects

    The same as in the Pods’ `spec.tolerations` parameter in Kubernetes;

    If the parameter is omitted or `false`, it will be determined automatically.
    - **settings.tolerations.effect**
      string
    - **settings.tolerations.key**
      string
    - **settings.tolerations.operator**
      string
    - **settings.tolerations.tolerationSeconds**
      integer

- **settings.tolerations.value**
  string

Cluster SLA Monitoring

> **i** The module lifecycle stage: General Availability

> **i** Web interfaces associated with the module: status, upmeter

The module tests the availability of the platform and the status of cluster components in real time, and outputs information in the form of corresponding dashboards.

Module features:

- continuous testing scenarios have been developed for all major cluster components;
- the test results are saved as metrics;
- displays a dashboard showing the functionality of the components;
- the data can be exported to any external Prometheus-compatible monitoring system.

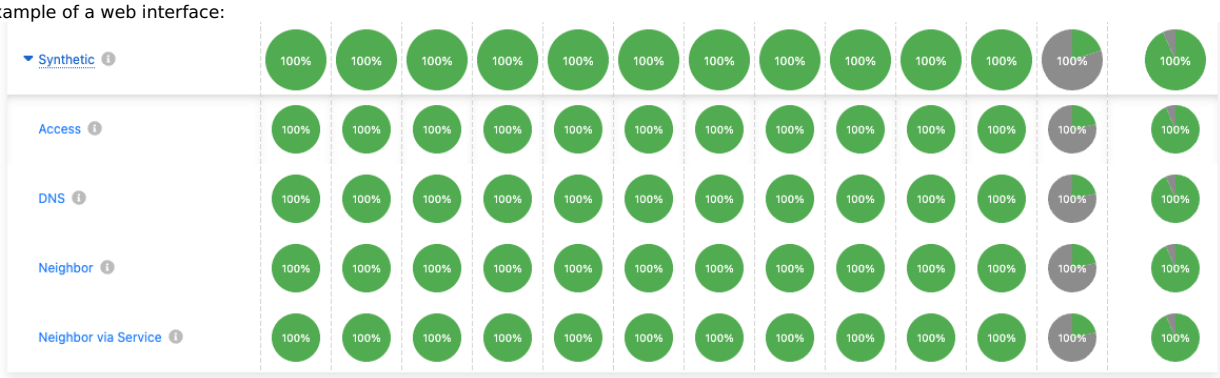
Module composition:

- **agent** — runs on master nodes, performs availability tests, and sends results to the server.
- **upmeter** — collects results and supports an API server to retrieve them.
- **front**
  - **status** — shows the availability level for the last 10 minutes (requires authorization, but it can be disabled);
  - **webui** — shows a dashboard with statistics on probes and availability groups (requires authorization).
- **smoke-mini** — supports continuous *smoke testing* using StatefulSet.

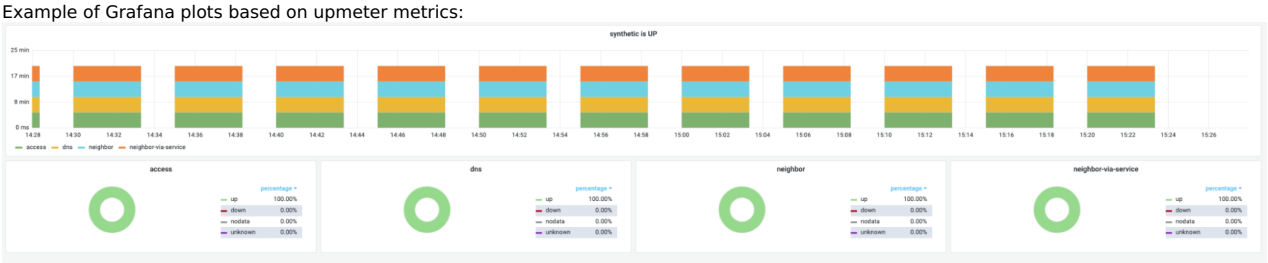
The module sends about 100 metric readings every 5 minutes. This figure depends on the number of Deckhouse Kubernetes Platform modules enabled.

Interface

Example of a web interface:



Example of Grafana plots based on upmeter metrics:



Configuration

> The module lifecycle stage: General Availability

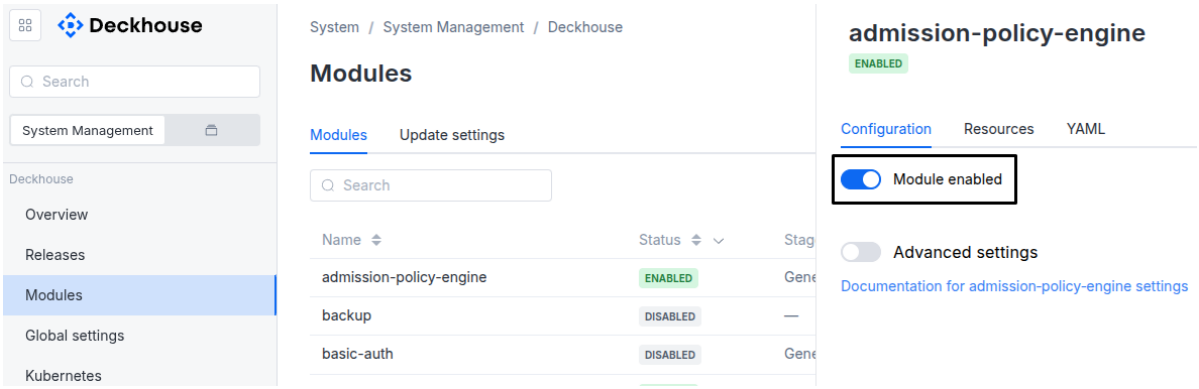
The module has 15 alerts.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `upmeter` module and enable (or disable) the “Module enabled” toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable upmeter
```

- **Using ModuleConfig `upmeter`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `upmeter` (create it if necessary);

  Example of a manifest to enable module `upmeter`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: upmeter
spec:
  enabled: true
```

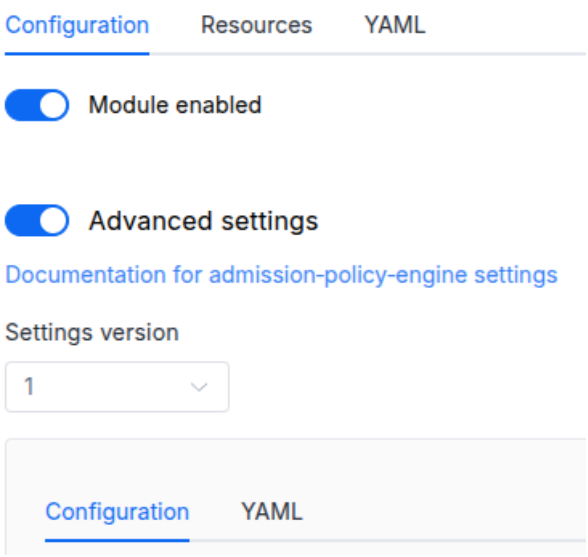
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `upmeter` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `upmeter` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `upmeter`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `upmeter` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc upmeter
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `upmeter` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `upmeter`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: upmeter
  spec:
    version: 3
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Conversion description is missing.

  Actions performed by the conversion (jq syntax)...

  - `del(.auth.status.password) | if .auth.status == {} then del(.auth.status) end | if .auth == {} then del(.auth) end`

    - `del(.auth.webui.password) | if .auth.webui == {} then del(.auth.webui) end | if .auth == {} then del(.auth) end`

- Updates from version **2** to **3**:

  Conversion description is missing.

  Actions performed by the conversion (jq syntax)...

    - `del(.smokeMini.ingressClass) | if .smokeMini == {} then del(.smokeMini) end`

Parameters

Schema version: 3

- **settings**
  object
    - **settings.auth**
      object

      Required value

      Configuration of authentication for Upmeter frontends.

        - **settings.auth.status**
          object

          Required value

          Authentication configuration.

            - **settings.auth.status.allowedUserEmails**
              array of strings

              An array of emails of users that can access module’s public web interfaces.

              This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

            - **settings.auth.status.allowedUserGroups**
              array of strings

              An array of user groups that can access Grafana & Prometheus.

              This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

              Caution! Note that you must add those groups to the appropriate field in the DexProvider config if this module is used together with the user-authn one.

            - **settings.auth.status.externalAuthentication**
              object

              Parameters to enable external authentication based on the Ingress NGINX external-auth mechanism that uses the NGINX auth_request module.

              > **i** External authentication is enabled automatically if the user-authn module is enabled.

                - **settings.auth.status.externalAuthentication.authSignInURL**
                  string

                  URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response code).

                - **settings.auth.status.externalAuthentication.authURL**
                  string

                  URL of the authentication service. If the user is authenticated, the service should return an HTTP 200 response code.

            - **settings.auth.status.whitelistSourceRanges**
              array of strings

              An array if CIDRs that are allowed to authenticate.

Example:

```
whitelistSourceRanges:
- 1.1.1.1/32
```

- **settings.auth.webui**
  object

  Required value

  Authentication configuration.

  - **settings.auth.webui.allowedUserEmails**
    array of strings

    An array of emails of users that can access module’s public web interfaces.

    This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

  - **settings.auth.webui.allowedUserGroups**
    array of strings

    An array of user groups that can access Grafana & Prometheus.

    This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

    Caution! Note that you must add those groups to the appropriate field in the DexProvider config if this module is used together with the user-authn one.

  - **settings.auth.webui.externalAuthentication**
    object

    Parameters to enable external authentication. Uses Ingress NGINX external-auth mechanism which is based on the the NGINX auth_request module.

    - **settings.auth.webui.externalAuthentication.authSignInURL**
      string

      URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response code).

    - **settings.auth.webui.externalAuthentication.authURL**
      string

      URL of the authentication service. If the user is authenticated, the service should return an HTTP 200 response code.

  - **settings.auth.webui.password**
    string

    Password for http authorization of the `admin` user. It is generated automatically, but you can change it.

    This parameter is used if the `externalAuthentication` is not enabled.

  - **settings.auth.webui.whitelistSourceRanges**
    array of strings

    An array if CIDRs that are allowed to authenticate.

    Example:

    ```
    whitelistSourceRanges:
    - 1.1.1.1/32
    ```

- **settings.disabledProbes**
  array of strings

  Group names or specific probes from a group. You can view the names in the web UI.

  For example:

```
disabledProbes:
  - "synthetic/api" # disable a specific probe
  - "synthetic/"   # disable a group of probes
  - control-plane  # / can be omitted
```

Default: `[]`

- **settings.https**
  object

  What certificate type to use with webui and status apps.

  This parameter completely overrides the `global.modules.https` settings.

  Examples:

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

  - **settings.https.certManager**
    object
    - **settings.https.certManager.clusterIssuerName**
      string

      What ClusterIssuer to use for webui/status.

      Currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available. Also, you can define your own.

      Default: `letsencrypt`

  - **settings.https.customCertificate**
    object
    - **settings.https.customCertificate.secretName**
      string

      The name of the secret in the `d8-system` namespace to use with webui/status.

      This secret must have the kubernetes.io/tls format.

      Default: `false`

  - **settings.https.mode**
    string

    The HTTPS usage mode:

    - `Disabled` — webui/status will work over HTTP only;
    - `CertManager` — webui/status will use HTTPS and get a certificate from the clusterissuer defined in the `certManager.clusterIssuerName` parameter.
    - `CustomCertificate` — webui/status will use HTTPS using the certificate from the `d8-system` namespace.
    - `OnlyInURI` — webui/status will work over HTTP (thinking that there is an external HTTPS load balancer in front that terminates HTTPS traffic). All the links in the `user-authn` will be generated using the HTTPS scheme. Load balancer should provide a redirect from HTTP to HTTPS.

    Default: `Disabled`

    Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.ingressClass**
  string

  The class of the Ingress controller used for the Upmeter.

Optional. By default, the `modules.ingressClass` global value is used

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.nodeSelector**
  object

  Node selector for Upmeter server. The same as in the Pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  nodeSelector:
    disktype: ssd
  ```

- **settings.smokeMini**
  object

  Configuration of authentication for smoke-mini.

  - **settings.smokeMini.auth**
    object

    Required value

    Authentication configuration.

    - **settings.smokeMini.auth.allowedUserGroups**
      array of strings

      An array of user groups that can access Grafana & Prometheus.

      This parameter is used if the `user-authn` module is enabled or the `externalAuthentication` parameter is set.

      Caution! Note that you must add those groups to the appropriate field in the DexProvider config if this module is used together with the user-authn one.

    - **settings.smokeMini.auth.externalAuthentication**
      object

      Parameters to enable external authentication. Uses Ingress NGINX external-auth mechanism which is based on the the NGINX auth_request module.

      - **settings.smokeMini.auth.externalAuthentication.authSignInURL**
        string

        URL to redirect the user for authentication (if the authentication service returned a non-200 HTTP response code).

      - **settings.smokeMini.auth.externalAuthentication.authURL**
        string

        URL of the authentication service. If the user is authenticated, the service should return an HTTP 200 response code.

    - **settings.smokeMini.auth.password**
      string

      Password for http authorization of the `admin` user. It is generated automatically, but you can change it.

      This parameter is used if the `externalAuthentication` is not enabled.

    - **settings.smokeMini.auth.whitelistSourceRanges**
      array of strings

      An array if CIDRs that are allowed to authenticate.

      Example:

      ```
      whitelistSourceRanges:
      - 1.1.1.1/32
      ```

- **settings.smokeMini.https**
  object

  What certificate type to use with smoke-mini.

  This parameter completely overrides the `global.modules.https` settings.

  Examples:

```
https:
  mode: CustomCertificate
  customCertificate:
    secretName: foobar
```

```
https:
  mode: CertManager
  certManager:
    clusterIssuerName: letsencrypt
```

  - **settings.smokeMini.https.certManager**
    object
    - **settings.smokeMini.https.certManager.clusterIssuerName**
      string

      What ClusterIssuer to use for smoke-mini.

      Currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available. Also, you can define your own.

      Default: `letsencrypt`

  - **settings.smokeMini.https.customCertificate**
    object
    - **settings.smokeMini.https.customCertificate.secretName**
      string

      The name of the secret in the `d8-system` namespace to use with smoke-mini.

      This secret must have the kubernetes.io/tls format.

      Default: `false`

  - **settings.smokeMini.https.mode**
    string

    The HTTPS usage mode:

    - `Disabled` — smoke-mini will work over HTTP only;
    - `CertManager` — smoke-mini will use HTTPS and get a certificate from the clusterissuer defined in the `certManager.clusterIssuerName` parameter.
    - `CustomCertificate` — smoke-mini will use HTTPS using the certificate from the `d8-system` namespace.
    - `OnlyInURI` — smoke-mini will work over HTTP (thinking that there is an external HTTPS load balancer in front that terminates HTTPS traffic). All the links in the `user-authn` will be generated using the HTTPS scheme. Load balancer should provide a redirect from HTTP to HTTPS.

    Default: `Disabled`

    Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.smokeMini.storageClass**
  string or boolean

  A StorageClass to use when checking the health of disks.

  If omitted, the StorageClass of the existing PVC is used. If there is no PVC yet, either the global StorageClass or `global.discovery.defaultStorageClass` is used, and if those are undefined, the emptyDir volume is used to store the data.

  `global.discovery.defaultStorageClass` is applied during module activation, changing default StorageClass in cluster won't result in disk re-provisioning.

Setting it to `false` forces the use of an emptyDir volume.

Default: `false`

Allowed values: `false`

Examples:

```
storageClass: false
```

```
storageClass: default
```

- **settings.smokeMiniDisabled**
  boolean

  Disables smokeMini.

  Disables “synthetic” probe group in Upmeter as well.

  Default: `false`

- **settings.statusPageAuthDisabled**
  boolean

  Disables authorization for the status domain.

  Default: `false`

- **settings.storageClass**
  string or boolean

  The name of the StorageClass to use for Upmeter storage (PVC).

  If omitted, the StorageClass of the existing PVC is used. If there is no PVC yet, the StorageClass will be used according to the global storageClass parameter setting.

  The global `storageClass` parameter is only considered when the module is enabled. Changing the global `storageClass` parameter while the module is enabled will not trigger disk re-provisioning.

  If `false` is specified, `emptyDir` will be forced to be used.

  Warning. Specifying a value different from the one currently used (in the existing PVC) will result in disk re-provisioning and all data will be deleted.

  Warning. When migrating Upmeter with local storage to other nodes, the pods will enter a Pending state. In this situation, it will be necessary to back up the Upmeter database, delete the old PVCs, and manually restart the pod. Here, local storage refers to a StorageClass associated not with network storage but with a local volume on a node (for example, a StorageClass created by the local-path-provisioner module).

  Allowed values: `false`

  Examples:

  ```
  storageClass: false
  ```

  ```
  storageClass: default
  ```

- **settings.tolerations**
  array of objects

  Node tolerations for Upmeter server. The same as in the Pods’ `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

```
tolerations:
- key: key1
  operator: Equal
  value: value1
  effect: NoSchedule
```

- **settings.tolerations.effect**
  string
- **settings.tolerations.key**
  string
- **settings.tolerations.operator**
  string
- **settings.tolerations.tolerationSeconds**
  integer
- **settings.tolerations.value**
  string

Authentication

user-authn module provides authentication by default. Also, externalAuthentication can be configured (see below). If these options are disabled, the module will use basic auth with the auto-generated password.

Use d8 k to see password:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values upmeter -o json | jq '.upmeter.internal.auth.webui.password'
```

Delete the Secret to re-generate password:

```
d8 k -n d8-upmeter delete secret/basic-auth-webui
```

Use d8 k to see password for status page:

```
d8 k -n d8-system exec svc/deckhouse-leader -c deckhouse -- deckhouse-controller module values upmeter -o json | jq '.upmeter.internal.auth.status.password'
```

Delete the Secret to re-generate password for status page:

```
d8 k -n d8-upmeter delete secret/basic-auth-status
```

> **Note!** The `auth.status.password` and `auth.webui.password` parameters are deprecated.

Custom Resources

> The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> The module lifecycle stage: General Availability

Downtime

Scope: Cluster
Version: v1alpha1

Downtime incident records

- **spec**
  array of objects

Required value

- **spec.affected**
  array of strings

  A list of affected groups.

- **spec.description**
  string

  Human readable incident information.

- **spec.endDate**
  string

  End of downtime (Unix time or RFC3339 date 2020-09-07T17:24:55Z).

- **spec.startDate**
  string

  Start of downtime (Unix time or RFC3339 date 2020-09-07T17:24:55Z).

- **spec.type**
  string

  Type of downtime incident.

  Allowed values: `Accident`, `Maintenance`, `InfrastructureMaintenance`, `InfrastructureAccident`

UpmeterHookProbe

Scope: Cluster
Version: v1

Upmeter CRD for deckhouse-configuration probe.

- **spec**
  object

  Required value

  - **spec.inited**
    string

    Required value

    The initial value set by upmeter agent

  - **spec.mirror**
    string

    Required value

    The mirrored value set by hook

UpmeterRemoteWrite

Scope: Cluster

- v1
- v1alpha1

Upmeter `remote_write` configuration.

- **spec**
  object

  Required value

  - **spec.additionalLabels**
    object

    Additional labels to add to all metrics.

  - **spec.config**
    object

Required value

Config defines the `remote_write` configuration for Prometheus.

- **spec.config.basicAuth**
  object

  BasicAuth for the URL.

  - **spec.config.basicAuth.password**
    string

    Required value

    The password for authentication.

  - **spec.config.basicAuth.username**
    string

    Required value

    The username for authentication.

- **spec.config.bearerToken**
  string

  Bearer token.

- **spec.config.headers**
  object

  Additional HTTP headers to send with each remote write request.

  - **spec.config.headers.X-Auth-Token**
    string

    The `X-Auth-Token` header specifying the authentication token.

  - **spec.config.headers.X-Scope-OrgID**
    string

    The `X-Scope-OrgID` header specifying the tenant ID.

- **spec.config.tlsConfig**
  object

  TLS Config to use for remote write.

  - **spec.config.tlsConfig.ca**
    string

    The CA certificate used when verifying the API server certificate.

    Default:

  - **spec.config.tlsConfig.insecureSkipVerify**
    boolean

    Disable target certificate validation.

    Default: `false`

- **spec.config.url**
  string

  Required value

  The URL of the endpoint to send samples to.

- **spec.intervalSeconds**
  number

  The size of timeslot to send.

  Default: `300`

  Allowed values: `30`, `300`

Deprecated resource. Support for the resource might be removed in a later release.

Upmeter `remote_write` configuration.

- **spec**
  object

  Required value

  - **spec.additionalLabels**
    object

    Additional labels to add to all metrics.

  - **spec.config**
    object

    Required value

    Config defines the `remote_write` configuration for Prometheus.

    - **spec.config.basicAuth**
      object

      BasicAuth for the URL.

      - **spec.config.basicAuth.password**
        string

        Required value

        The password for authentication.

      - **spec.config.basicAuth.username**
        string

        Required value

        The username for authentication.

    - **spec.config.bearerToken**
      string

      Bearer token.

    - **spec.config.headers**
      object

      Additional HTTP headers to send with each remote write request.

      - **spec.config.headers.X-Auth-Token**
        string

        The `X-Auth-Token` header specifying the authentication token.

      - **spec.config.headers.X-Scope-OrgID**
        string

        The `X-Scope-OrgID` header specifying the tenant ID.

    - **spec.config.tlsConfig**
      object

      TLS Config to use for remote write.

      - **spec.config.tlsConfig.ca**
        string

        The CA certificate used when verifying the API server certificate.

        Default:

      - **spec.config.tlsConfig.insecureSkipVerify**
        boolean

        Disable target certificate validation.

        Default: `false`

    - **spec.config.url**
      string

Required value

The URL of the endpoint to send samples to.

- **spec.intervalSeconds**
  number

  The size of timeslot to send.

  Default: `300`

  Allowed values: `30`, `300`

FAQ

> **i** The module lifecycle stage: General Availability

Why can't `upmeter-probe-*` pods be placed in the cluster, why are some pods constantly deleted?

The module implements tests for the availability of the functionality of some Kubernetes controllers.

The tests are performed by creating and deleting temporary pods.

`upmeter-probe-scheduler` objects implement a test for the functionality of placing pods on nodes. As part of the test, a Pod is created and placed on a node. Then this pod is deleted.

`upmeter-probe-controller-manager` objects implement a test for the health of `kube-controller-manager`.

`StatefulSet` is created for the test, and it is checked that this object has spawned a Pod (since the actual placement of the pod is not required and is checked as part of another test, a Pod is created that is guaranteed not to be placed, i.e. is in the `Pending` state). Deletes the StatefulSet, checks that the Pod it spawned has been deleted.

`smoke-mini` objects implement network connectivity testing between nodes. Five `StatefulSet` with one replica are deployed for testing. The test checks connectivity between `smoke-mini` Pods as well as network connectivity with `upmeter-agent` Pods running on master nodes.
Once per minute, one of the `smoke-mini` Pods is migrated to another node.

Examples

> **i** The module lifecycle stage: General Availability

An example of the `UpmeterRemoteWrite` configuration

```
apiVersion: deckhouse.io/v1
kind: UpmeterRemoteWrite
metadata:
  labels:
    heritage: upmeter
    module: upmeter
  name: victoriametrics
spec:
  additionalLabels:
    cluster: cluster-name
    some: fun
  config:
    url: https://upmeter-victoriametrics.whatever/api/v1/write
    basicAuth:
      password: "Cdp#Cd.OxfZsx4*89SZ"
      username: upmeter
  intervalSeconds: 300
```

The user-authn module

> **i** The module lifecycle stage: General Availability

> **i** Web interfaces associated with the module: kubeconfig

The module sets up a unified authentication system integrated with Kubernetes and Web interfaces used in other modules (Grafana, Dashboard, etc.).

It consists of the following components:

- dex — is a federated OpenID Connect provider that acts as an identity service for static users and can be linked to one or more ID providers (e.g., SAML providers, GitHub, and Gitlab);
- `kubeconfig-generator` (in fact, dex-k8s-authenticator) — is a helper web application that (being authorized with dex) generates kubectl commands for creating and modifying a kubeconfig;
- `dex-authenticator` (in fact, `oauth2-proxy`) is an application that receives Ingress NGINX requests (via the `auth_request` module) and authenticates them with Dex.

Management of static users is performed using the User and Group resources:

- The User object stores user information, including the email address and an encrypted password hash (the plain-text password is not stored);
- The Group object defines a list of users grouped together for easier access management.

The following external authentication protocols/providers are supported:

- GitHub;
- GitLab;
- BitBucket Cloud;
- Crowd;
- LDAP;
- OIDC.

You can use several external authentication providers simultaneously.

Integration features

Basic authentication for the Kubernetes API requests

Basic authentication for the Kubernetes API requests is currently only available for the Crowd provider (with the `enableBasicAuth` parameter).

> **i** You can also connect to the Kubernetes API via other supported external providers.

Application integration

To enable authentication for any web application running in Kubernetes, create a *DexAuthenticator* resource in the application's *Namespace* and add several annotations to the *Ingress* resource. This will enable you to:

- limit the list of groups with access;
- limit the list of addresses for which authentication is allowed;
- integrate the application into a unified authentication system if the application supports OIDC. For that, Kubernetes will create a resource *DexClient* in the application *Namespace*. A secret with data for connecting to Dex via OIDC will also be created in that *Namespace*.

Following such an integration, you can:

- limit the list of groups for which the connection is allowed;
- define a list of clients with trusted OIDC tokens (`trustedPeers`).

Web interface for generating ready-made kubeconfig files

The module allows you to automatically generate configuration for kubectl or other Kubernetes tools.

The user will be presented with a set of commands to configure kubectl once authorized in the

generator's web interface. These commands can be copied and pasted into the console to use kubectl. The authentication mechanism for kubeconfig uses an OIDC token. The OIDC session can be extended automatically if the authentication provider used in Dex supports session extension. For this, the `refresh token` is specified in kubeconfig.

On top of that, you can configure multiple `kube-apiserver` addresses and CA certificates for each of them. This may come in handy, e. g., if access to the Kubernetes cluster is via VPN or direct connection.

Exposing the Kubernetes API over Ingress

The kube-apiserver component (without advanced configuration) is only accessible in the internal cluster network. This module enables easy and secure access to Kubernetes API from outside the cluster. The API server is exposed on a dedicated domain (for more details, see the section on service domains in the documentation).

When configuring, you can:

- list network addresses from which connection is allowed;
- list groups that are allowed to access the API server;
- specify Ingress-controller to authenticate on.

By default, a special CA certificate will be generated and the kubeconfig generator will be automatically configured.

Extensions by Flant

The module uses a modified version of Dex to support:

- groups for static user accounts and Bitbucket Cloud provider (parameter `bitbucketCloud`);
- passing the `group` parameter to clients;
- the `obsolete tokens` mechanism to avoid a race condition when an OIDC client renews a token.
- Kerberos (SPNEGO) support for the LDAP connector. When enabled, Dex accepts `Authorization: Negotiate` tickets, validates them with a service keytab, and completes the login without rendering the password form.

High availability mode

The module also supports the `highAvailability` mode. When this mode is enabled, all components responding to the `auth requests` are deployed with the redundancy required to operate continuously without failure. Thus, the user authentication sessions are kept alive even if any of the authenticator instances fail.

Third-party components

List of third-party software used in the `user-authn` module:

- Dex 2.44.0

  License: Apache License 2.0

  A Federated OpenID Connect Provider with pluggable connectors.

- OAuth2 Proxy 7.5.1

  License: MIT License

  A reverse proxy and static file server that provides authentication using Providers (Google, GitHub, and others) to validate accounts by email, domain or group.

Configuration

> **i** The module lifecycle stage: General Availability

The module has 3 alerts.

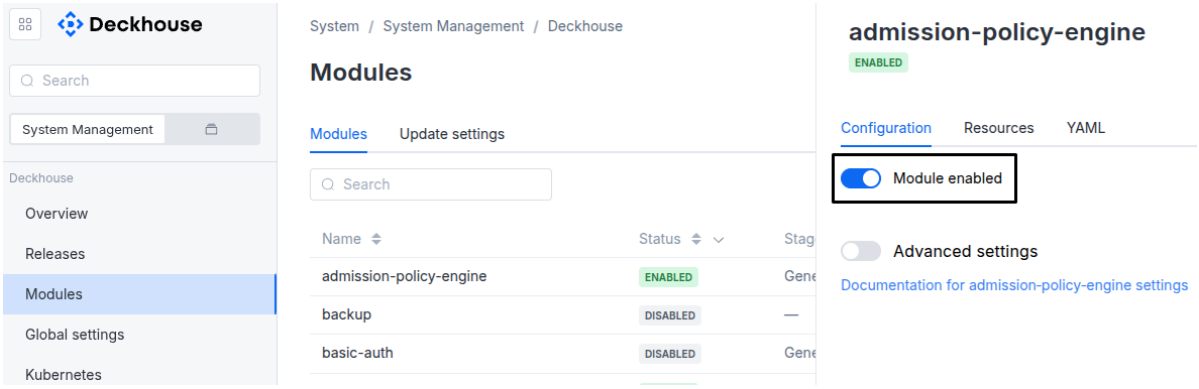
The module is **enabled** by default in the `Default` bundle. The module is **disabled** by default in the following bundles: `Managed`, `Minimal`.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `user-authn` module and enable (or disable) the "Module enabled" toggle. Save

changes.

Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

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

- **Using ModuleConfig `user-authn`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `user-authn` (create it if necessary);

  Example of a manifest to enable module `user-authn`:

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

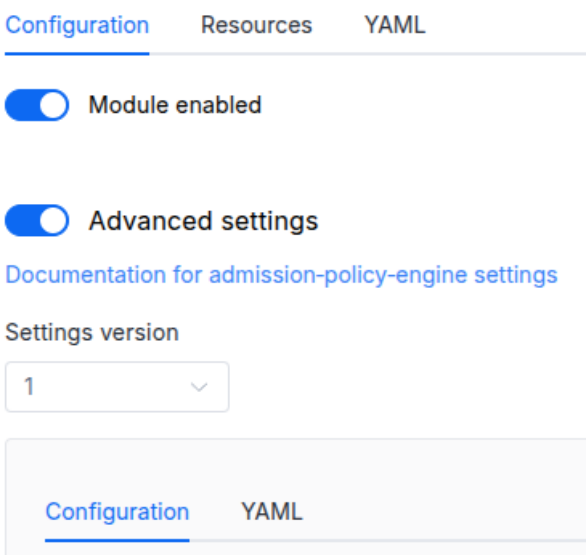
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `user-authn` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `user-authn` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `user-authn`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `user-authn` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

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

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `user-authn` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `user-authn`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: user-authn
  spec:
    version: 2
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Conversions

The module is configured using the ModuleConfig resource, the schema of which contains a version number. When you apply an *old* version of the ModuleConfig schema in a cluster, automatic transformations are performed. To manually update the ModuleConfig schema version, the following steps must be completed **sequentially** for each version :

- Updates from version **1** to **2**:

  Replace `publishAPI.enable` with `publishAPI.enabled`.

Parameters

Schema version: 2

- **settings**
  object
  - **settings.controlPlaneConfigurator**
    object

    Parameters of the `control-plane-manager` module for automatic configuration of `kube-apiserver`.

    - **settings.controlPlaneConfigurator.dexCAMode**
      string

      How to determine the CA that will be used when configuring `kube-apiserver`.

      - `Custom` — use the CA explicitly set via the `dexCustomCA` parameter (see below). This option comes in handy if you use an external HTTPS load balancer in front of Ingresses, and this load balancer relies on a self-signed certificate.
      - `DoNotNeed` — a CA is not required (e.g., when using a public LE or other TLS providers).
      - `FromIngressSecret` — extract the CA of certificate from the Secret that is used in the Ingress. This option comes in handy if you use self-signed certificates with Ingresses.

      Default: `DoNotNeed`

      Allowed values: `Custom`, `DoNotNeed`, `FromIngressSecret`

    - **settings.controlPlaneConfigurator.dexCustomCA**
      string

      The CA to use if `dexCAMode` is `Custom`. Plain text (no base64).

    - **settings.controlPlaneConfigurator.enabled**
      boolean

      Defines if the `control-plane-manager` module should be used to configure OIDC for the `kube-apiserver`.

      Default: `true`

  - **settings.highAvailability**
    boolean

    Manually enable the high availability mode.

    By default, Deckhouse automatically decides whether to enable the HA mode. Click here to learn more about the HA mode for modules.

    Examples:

    ```
    highAvailability: true
    ```

    ```
    highAvailability: false
    ```

  - **settings.https**
    object

    What certificate type to use with Dex/kubeconfig-generator.

    This parameter completely overrides the `global.modules.https` settings.

    Examples:

    ```
    https:
      mode: CustomCertificate
      customCertificate:
        secretName: foobar
    ```

    ```
    https:
      mode: CertManager
      certManager:
        clusterIssuerName: letsencrypt
    ```

- **settings.https.certManager**
  object
  - **settings.https.certManager.clusterIssuerName**
    string

    What ClusterIssuer to use for Dex/kubeconfig-generator.

    Currently, `letsencrypt`, `letsencrypt-staging`, `selfsigned` are available. Also, you can define your own.

    Default: `letsencrypt`

- **settings.https.customCertificate**
  object
  - **settings.https.customCertificate.secretName**
    string

    The name of the Secret in the `d8-system` namespace to use with Dex/kubeconfig-generator.

    This Secret must have the `kubernetes.io/tls` format.

    Default: `false`

- **settings.https.mode**
  string

  The HTTPS usage mode:

  - `CertManager` — Dex/kubeconfig-generator will use HTTPS and get a certificate from the ClusterIssuer defined in the `certManager.clusterIssuerName` parameter.
  - `CustomCertificate` — Dex/kubeconfig-generator will use HTTPS using the certificate from the `d8-system` namespace.
  - `Disabled` — Dex/kubeconfig-generator will work over HTTP only;
  - `OnlyInURI` — Dex/kubeconfig-generator will work over HTTP (thinking that there is an external HTTPS load balancer in front that terminates HTTPS traffic). All the links in the `user-authn` will be generated using the HTTPS scheme. Load balancer should provide a redirect from HTTP to HTTPS.

  Default: `Disabled`

  Allowed values: `Disabled`, `CertManager`, `CustomCertificate`, `OnlyInURI`

- **settings.idTokenTTL**
  string

  TTL of the ID token.

  Set as a string specifying hours, minutes, or seconds: `30m`, `20s`, `2h30m10s`, `24h`.

  Defines the minimum possible user session lifetime (via `keepUsersLoggedInFor` in the DexAuthenticator resource and parameters like `auth.sessionTTL`, for example, in the `console` module).

  If the session TTL is less than or equal to `idTokenTTL`, the effective logout time is `idTokenTTL + 1s`.

  To force logout sooner than the default `idTokenTTL`, decrease this parameter value as well.

  Default: `10m`

  Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

- **settings.ingressClass**
  string

  The class of the Ingress controller that will be used for Dex/kubeconfig-generator.

  An optional parameter; by default, the `modules.ingressClass` global value is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.kubeconfigGenerator**
  array of objects

  An array in which additional possible methods for accessing the API server are specified.

This option comes in handy if you prefer not to grant access to the cluster's API via Ingress but rather do it by other means (e.g., using a bastion host or over OpenVPN).

- **settings.kubeconfigGenerator.description**
  string

  A couple of words how this authentication method differs from others.

- **settings.kubeconfigGenerator.id**
  string

  Required value

  The name of the method for accessing the API server (no spaces, lowercase letters).

  Pattern: `^[\@\.\:0-9a-z._-]+$`

- **settings.kubeconfigGenerator.masterCA**
  string

  A CA for accessing the API:

  - If the parameter is not set, Kubernetes CA is used.
  - We recommend using a self-signed certificate (and specify it as masterCA) if an HTTP proxy (that terminates HTTPS traffic) is used for exposing.
- **settings.kubeconfigGenerator.masterURI**
  string

  Required value

  If you plan to use a TCP proxy, then you must configure a certificate on the API server's side for the TCP proxy address. Suppose your API servers use three different addresses (`192.168.0.10`, `192.168.0.11`, and `192.168.0.12`) while the client uses a TCP load balancer (say, `192.168.0.15`). In this case, you have to re-generate the API server certificates:

  - edit `kubeadm-config`: `d8 k -n kube-system edit configmap kubeadm-config` and add `192.168.0.15` to `.apiServer.certSANs`;
  - save the resulting config: `kubeadm config view > kubeadmconf.yaml`;
  - delete old API server certificates: `mv /etc/kubernetes/pki/apiserver.* /tmp/`;
  - reissue new certificates: `kubeadm init phase certs apiserver --config=kubeadmconf.yaml`;
  - restart the API server's container: `docker ps -a | grep 'kube-apiserver' | grep -v pause| awk '{print $1}' | xargs docker restart`;
  - repeat this step for all master nodes.

- **settings.nodeSelector**
  object

  The same as in the Pods' `spec.nodeSelector` parameter in Kubernetes.

  If the parameter is omitted or `false`, it will be determined automatically.

- **settings.passwordPolicy**
  object

  Default:

  - **settings.passwordPolicy.complexityLevel**
    string

    Password complexity level.

    Depending on the complexity level, a password must meet the following minimum requirements:

    - `None`: 1 character.
    - `Low`: 8 characters.
    - `Fair`: 8 characters, 1 uppercase letter (`A`–`Z`), 1 lowercase letter (`a`–`z`), 1 digit.
    - `Good`: 8 characters, 1 uppercase letter (`A`–`Z`), 1 lowercase letter (`a`–`z`), 1 digit, 1 special character (`!@#$%^&*`).
    - `Excellent`: 8 characters, 1 uppercase letter (`A`–`Z`), 1 lowercase letter (`a`–`z`), 1 digit, 1 special character (`!@#$%^&*`), no more than 2 identical characters in a row.

    Default: `Fair`

    Allowed values: `None`, `Low`, `Fair`, `Good`, `Excellent`

- **settings.passwordPolicy.lockout**
  object

  Temporary user lockout settings after consecutive failed login attempts.

  - **settings.passwordPolicy.lockout.lockDuration**
    string

    Required value

    Temporary lockout duration. Can be set in seconds, minutes, hours, or days: `10s`, `5m`, `1h`, `3d`.

    Pattern: `^(?:(?:[1-9]|[1-9][0-9]+)d)?(?:(?:[1-9]|[1-9][0-9]+)h)?(?:(?:[1-9]|[1-9][0-9]+)m)?(?:(?:[1-9]|[1-9][0-9]+)s)?$`

  - **settings.passwordPolicy.lockout.maxAttempts**
    integer

    Required value

    Number of consecutive failed login attempts after which the user will be temporarily locked out.

- **settings.passwordPolicy.passwordHistoryLimit**
  integer

  Number of the user’s previous passwords stored in history to prevent their reuse.

  Default: `5`

  Allowed values: `0 <= X`

- **settings.passwordPolicy.rotation**
  object

  Password rotation settings.

  - **settings.passwordPolicy.rotation.interval**
    string

    Required value

    Time interval after which the password must be changed. Can be set in seconds, minutes, hours or days: `10s`, `5m`, `1h`, `3d`.

    Pattern: `^(?:(?:[1-9]|[1-9][0-9]+)d)?(?:(?:[1-9]|[1-9][0-9]+)h)?(?:(?:[1-9]|[1-9][0-9]+)m)?(?:(?:[1-9]|[1-9][0-9]+)s)?$`

- **settings.publishAPI**
  object

  Settings for exposing the API server using Ingress.

  - **settings.publishAPI.addKubeconfigGeneratorEntry**
    boolean

    Setting it to `false` will remove an entry in kubeconfig-generator.

    Default: `true`

  - **settings.publishAPI.enabled**
    boolean

    Setting it to `true` will create an Ingress resource in the `d8-user-authn` namespace in the cluster (it exposes the Kubernetes API).

    Default: `false`

  - **settings.publishAPI.https**
    object

    The HTTPS mode for the API server Ingress.

    Examples:

```
https:
  mode: SelfSigned
```

```
https:
  mode: Global
  global:
    kubeconfigGeneratorMasterCA: plainstring
```

- **settings.publishAPI.https.global**
  object

  An additional parameter for the `Global` mode.

  - **settings.publishAPI.https.global.kubeconfigGeneratorMasterCA**
    string

    If there is an external load balancer in front of the Ingress that terminates HTTPS traffic using non-public CA, then you need to specify the CA so it will be included in kubectl-config.

    If you are using certificates issued by the `cert-manager` module and Let's Encrypt in your cluster, you should set an empty string `""` as the value.

    Also, you can set the external LB's certificate itself as a CA if you can't get the CA that signed it for some reason. Note that after the certificate is updated on the LB, all the previously generated kubeconfigs will stop working.

- **settings.publishAPI.https.mode**
  string

  The mode of issuing certificates for the Ingress resource.

  In the `SelfSigned` mode, a CA-signed certificate will be issued for the Ingress resource.

  Use the following command to get the certificate: `d8 k -n d8-user-authn get secrets kubernetes-api-ca-key-pair -oyaml`.

  In the `Global` mode, the policies specified in the `global.modules.https.mode` global parameter will be applied. Thus, if the global parameter has the `CertManager` mode set (with `letsencrypt` as the ClusterIssuer), then the Let's Encrypt certificate will be issued for the Ingress resource.

  Default: `SelfSigned`

  Allowed values: `SelfSigned`, `Global`

- **settings.publishAPI.ingressClass**
  string

  The Ingress class that will be used to expose the Kubernetes API via Ingress.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **settings.publishAPI.whitelistSourceRanges**
  array of strings

  An array of CIDRs that are allowed to connect to the API server.

  - **Element of the array**
    string

    Pattern: `^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\/(3[0-2]|[1-2][0-9]|[0-9]))?$`

- **settings.refreshTokenAbsoluteLifetime**
  string

  Maximum lifetime of refresh tokens (maps to Dex `expiry.refreshTokens.absoluteLifetime`).

  This setting controls how long user sessions can persist. After this period expires, users will need to re-authenticate, even if they have active refresh tokens.

It is specified as a string containing the time unit in hours, minutes and seconds: 30m, 20s, 2h30m10s, 24h.

Default: `4380h`

Pattern: `^([0-9]+h)?([0-9]+m)?([0-9]+s)?$`

- **settings.staticUsers2FA**
  object
  - **settings.staticUsers2FA.enabled**
    boolean

    Required value

    If set to `true`, the static users will be required to use two-factor authentication (2FA) when logging in. This option is useful for enhancing security by requiring an additional verification step during the login process.

    Default: `false`

  - **settings.staticUsers2FA.issuerName**
    string

    The issuer name for the two-factor authentication (2FA) tokens. This name is visible to users in the 2FA application (e.g., Google Authenticator).

    Default: `Deckhouse`

- **settings.tolerations**
  array of objects

  The same as in the Pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  Example:

  ```
  tolerations:
  - key: key1
    operator: Equal
    value: value1
    effect: NoSchedule
  ```

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

The creation of the `DexAuthenticator` Custom Resource leads to the automatic deployment of oauth2-proxy to your application's namespace and connecting it to Dex.

Caution! Since using OpenID Connect over HTTP poses a significant threat to security (the fact that Kubernetes API server doesn't support OICD over HTTP confirms that), this module can only be installed if HTTPS is enabled (to do this, set the `https.mode` parameter to the value other than `Disabled` either at the cluster level or in the module).

Caution! When this module is enabled, authentication in all web interfaces will be switched from HTTP Basic Auth to Dex (the latter, in turn, will use the external providers that you have defined). To configure kubectl, go to `https://kubeconfig.<modules.publicDomainTemplate>/`, log in to your external provider's account and copy the shell commands to your console.

Caution! The API server requires additional configuration to use authentication for dashboard and kubectl. The control-plane-manager module (enabled by default) automates this process.

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

DexAuthenticator

Short names: dex-auth

Scope: Namespaced

- v1
- v2alpha1
- v1alpha1

After the `DexAuthenticator` object appears in the namespace, the following objects will be created:

- Deployment containing OAuth2-Proxy and Redis containers;
- Service, pointing to OAuth2-Proxy;
- Ingress resource, configured to receive requests on `https://<applicationDomain>/dex-authenticator` and send it to a service side;
- HTTPRoute resource (optional), configured to receive requests on `https://<applicationDomain>/dex-authenticator` and send it to a service side;
- Secrets, needed to access Dex.

Warning. DexAuthenticator only works with HTTPS. The application Ingress must have TLS configured.

Warning. After restarting a pod with an OAuth2-Proxy, the current Access Token and ID Token will be queried (using the refresh token) and stored in a Redis memory.

Example:

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  name: app-name
  namespace: app-namespace
spec:
  applicationDomain: app-name.kube.my-domain.com
  sendAuthorizationHeader: false
  applicationIngressCertificateSecretName: ingress-tls
  applicationIngressClassName: nginx
  keepUsersLoggedInFor: 720h
  allowedGroups:
  - everyone
  - admins
  whitelistSourceRanges:
  - 1.1.1.1/32
  - 192.168.0.0/24
  additionalApplications:
  - domain: additional-app-name.kube.my-domain.com
    ingressSecretName: ingress-tls
    ingressClassName: nginx
    signOutURL: "/logout"
    whitelistSourceRanges:
    - 2.2.2.2/32
```

- **spec**
  object

  Required value

  - **spec.additionalApplications**
    array of objects

A list of additional applications for which user authentication is required.

- **spec.additionalApplications.domain**
  string

  Required value

  An application domain from which the user request will be redirected for authentication in Dex.

  > **Warning.** Specify a DNS name that complies with RFC 1123 (subdomain) and does not include an HTTP scheme. IP addresses are not allowed.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **spec.additionalApplications.ingressClassName**
  string

  Required value

  The name of the Ingress class to be used in the Ingress resource (it must match the Ingress class name for the application domain).

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **spec.additionalApplications.ingressSecretName**
  string

  The name of the Secret containing the TLS certificate (from the domain) used in the applications Ingress resource. The Secret must be in the same namespace as DexAuthenticator.

  Pattern: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

- **spec.additionalApplications.signOutURL**
  string

  The URL of the application from which requests will be redirected to the sign-out URL of the `dex-authenticator`.

  It is used in the application to direct logout requests. A separate Ingress resource will be created for the specified URL, and requests to it will be redirected to `dex-authenticator`.

  The value must not contain line breaks.

  Pattern: `^[^\r\n]*$`

  Maximum length: `2048`

- **spec.additionalApplications.whitelistSourceRanges**
  array of strings

  CIDRs that are allowed to authenticate. Authentication is allowed without IP address restrictions, If not specified.

  Example:

  ```
  whitelistSourceRanges:
  - 192.168.42.0/24
  ```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **spec.allowedEmails**
  array of strings

  A list of emails of users are allowed to connect to the client.

  By default, all users can connect.

  Values must not contain line breaks.

Default: `All emails are allowed.`

- **Element of the array**
  string

  Pattern: `^[^\r\n]*$`

  Maximum length: `320`

- **spec.allowedGroups**
  array of strings

  A list of groups whose members are allowed to connect to the client.

  By default, all groups can connect.

  Values must not contain line breaks.

  Default: `All groups are allowed.`

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `512`

- **spec.applicationDomain**
  string

  Required value

  An address for the application (DNS domain for Ingress resource) from which the user request will be redirected for authentication in Dex.

  > **Warning.** Specify a DNS name that complies with RFC 1123 (subdomain) and does not include an HTTP scheme. IP addresses are not allowed.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  applicationDomain: my-app.domain.com
  ```

- **spec.applicationIngressCertificateSecretName**
  string

  The name of the Secret containing the TLS certificate for the application domain (is used in the Ingress resource). The Secret must be located in the same namespace as the DexAuthenticator.

  Pattern: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

  Example:

  ```
  applicationIngressCertificateSecretName: ingress-tls
  ```

- **spec.applicationIngressClassName**
  string

  Required value

  The name of the Ingress class to be used in the Ingress resource (it must match the Ingress class name for the application domain).

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  applicationIngressClassName: nginx
  ```

- **spec.gatewayAPI**
  object

A set of parameters to use in HTTPRoute object.

> **Warning.** Only applicable in clusters with Gateway API support.

> **Warning.** Define the same ListenerSet object which is used to publish the application domain.

- **spec.gatewayAPI.applicationHTTPRouteListenerSetName**
  string

  Required value

  Name of the ListenerSet object from the Gateway API to be referred to from the HTTPRoute.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Maximum length: `253`

  Example:

  ```
  applicationHTTPRouteListenerSetName: default-listenerset
  ```

- **spec.gatewayAPI.applicationHTTPRouteListenerSetNamespace**
  string

  Namespace where the ListenerSet object from the Gateway API is deployed to be referred to from the HTTPRoute.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Maximum length: `63`

  Example:

  ```
  applicationHTTPRouteListenerSetNamespace: default
  ```

- **spec.gatewayAPI.applicationHTTPRouteListenerSetSectionName**
  string

  Section name from the ListenerSet object in the Gateway API the HTTPRoute refers to.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Maximum length: `253`

  Example:

  ```
  applicationHTTPRouteListenerSetSectionName: default-listener
  ```

- **spec.highAvailability**
  boolean

  Enables high availability mode for `dex-authenticator`.

  When this mode is enabled, multiple pod replicas will be deployed to ensure fault tolerance.

  Default: `false`

- **spec.keepUsersLoggedInFor**
  string

  Maximum period of user inactivity during which the user session remains active.

  Set as a string specifying hours, minutes, or seconds: `30m`, `20s`, `2h30m10s`, `24h`.

  Specified value can’t be shorter or equal to the `idTokenTTL` value set in the module settings. If you set a value that is shorter or equal to `idTokenTTL`, the effective logout time will be `idTokenTTL + 1s`.

  To force logout sooner than the default `idTokenTTL`, decrease the `idTokenTTL` parameter

value in the module settings as well.

Default: `168h`

Example:

```
keepUsersLoggedInFor: 24h
```

- **spec.nodeSelector**
  object

  If specified, the `dex-authenticator` pods nodeSelector.

  Format: the standard `nodeSelector` list. Instance pods inherit this field as is.

- **spec.resources**
  object

  Container resources (CPU/memory) — requests/limits.

  If this field is set, the specified values are applied and VPA is disabled for this instance. If the field is not set, behavior remains the same: when the VPA module is enabled, resources are managed by VPA, when it is disabled, default values are used.

  Resources can be specified either directly for the `dex-authenticator` container (for backward compatibility) or in nested objects — separately for each container.

  - **spec.resources.limits**
    object

    Upper limit of resources for the `dex-authenticator` container (CPU/memory). This field is kept for backward compatibility.

  - **spec.resources.redis**
    object

    Resources of the `redis` container (CPU/memory) — requests/limits.

    - **spec.resources.redis.limits**
      object

      Upper limit of resources for the `redis` container (CPU/memory)

    - **spec.resources.redis.requests**
      object

      Guaranteed minimum resources for the `redis` container (CPU/memory).

  - **spec.resources.requests**
    object

    Guaranteed minimum resources for the `dex-authenticator` container (CPU/memory). This field is kept for backward compatibility.

- **spec.sendAuthorizationHeader**
  boolean

  Request to application will be sent with `Authorization: Bearer` header when the option is switched to true.

- **spec.signOutURL**
  string

  The URL of the application from which requests will be redirected to the sign-out URL of the `dex-authenticator`.

  It is used in the application to direct logout requests. A separate Ingress resource will be created for the specified URL, and requests to it will be redirected to `dex-authenticator`.

  The value must not contain line breaks.

  Pattern: `^[^\r\n]*$`

  Maximum length: `2048`

- **spec.tolerations**
  array of objects

If specified the `dex-authenticator` pods tolerations.

Format: the standard toleration list. Instance pods inherit this field as is.

- **spec.tolerations.effect**
  string

  Defines the taint's effect this toleration is associated with (it matches any effect if empty).

  Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

- **spec.tolerations.key**
  string

  Defines the taint's key this toleration is associated with (it matches any key if empty).

  If the key is omitted (emplty), then `operator` must be set to `Exists` (which means it matches any `value` and `key`).

- **spec.tolerations.operator**
  string

  Defines how the key and its value are linked — whether the key must be `Equal` to the value or can have any value.

  Setting it to `Exists` is equivalent to the value having any value so that the pod with the specified toleration can match the corresponding taint.

  Default: `Equal`

  Allowed values: `Exists`, `Equal`

- **spec.tolerations.tolerationSeconds**
  integer

  Specifies the period (in seconds) that the scheduler waits before evicting a Pod from a node if the toleration no longer matches the taint (**only** applicable to the `NoExecute` effect, otherwise ignored).

  If the parameter **is not set**, the Pod will not be evicted from the node if the toleration no longer matches the taint. If the parameter **is set to zero** (or negative), the Pod will be evicted immediately if the toleration no longer matches the taint.

  The default is: not set.

- **spec.tolerations.value**
  string

  The value that the toleration must match.

  Must be empty if the `operator` parameter is set to `Exists`.

- **spec.whitelistSourceRanges**
  array of strings

  CIDRs that are allowed to authenticate. Authentication is allowed without IP address restrictions, If not specified.

  Example:

  ```
  whitelistSourceRanges:
  - 192.168.42.0/24
  ```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **spec**
  object

  Required value

  - **spec.allowedEmails**
    array of strings

A list of emails of users are allowed to connect to the client.

By default, all users can connect.

Values must not contain line breaks.

Default: `All users are allowed.`

- **Element of the array**
  string

  Pattern: `^[^\r\n]*$`

  Maximum length: `320`

- **spec.allowedGroups**
  array of strings

  A list of groups whose members are allowed to connect to the client.

  By default, all groups can connect.

  Values must not contain line breaks.

  Default: `All groups are allowed.`

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `512`

- **spec.applications**
  array of objects

  Required value

  A list of applications for which user authentication is required.

  - **spec.applications.domain**
    string

    Required value

    An application domain (for Ingress resources) from which the user request will be redirected for authentication in Dex.

    > **Warning.** Specify a DNS name that complies with RFC 1123 (subdomain) and does not include an HTTP scheme. IP addresses are not allowed.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  - **spec.applications.gatewayAPI**
    object

    A set of parameters to use in HTTPRoute object.

    > **Warning.** Only applicable in clusters with Gateway API support.

    > **Warning.** Define the same ListenerSet object which is used to publish the application domain.

    - **spec.applications.gatewayAPI.httpRouteListenerSetName**
      string

      Required value

      Name of the ListenerSet object from the Gateway API to be referred to from the HTTPRoute.

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-`

9])?)*$`

Maximum length: `253`

Example:

```
httpRouteListenerSetName: default-listener-set
```

- **spec.applications.gatewayAPI.httpRouteListenerSetNamespace**
  string

  Namespace where the ListenerSet object from the Gateway API is deployed to be referred to from the HTTPRoute.

  If not set, the namespace of the `DeAuthenticator` is used.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Maximum length: `63`

  Example:

  ```
  httpRouteListenerSetNamespace: default
  ```

- **spec.applications.gatewayAPI.httpRouteListenerSetSectionName**
  string

  Section name from the ListenerSet object in the Gateway API the HTTPRoute refers to.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Maximum length: `253`

  Example:

  ```
  httpRouteListenerSetSectionName: web-listener
  ```

- **spec.applications.ingressClassName**
  string

  Required value

  The name of the Ingress class to be used in the Ingress resource (it must match the Ingress class name for the application domain).

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **spec.applications.ingressSecretName**
  string

  The name of the Secret containing the TLS certificate for the application domain (is used in the Ingress resource). The Secret must be located in the same namespace as the DexAuthenticator.

  Pattern: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

- **spec.applications.signOutURL**
  string

  The URL of the application from which requests will be redirected to the sign-out URL of the `dex-authenticator`.

  It is used in the application to direct logout requests. A separate Ingress resource will be created for the specified URL, and requests to it will be redirected to `dex-authenticator`.

  The value must not contain line breaks.

  Pattern: `^[^\r\n]*$`

  Maximum length: `2048`

- **spec.applications.whitelistSourceRanges**
  array of strings

  CIDRs that are allowed to authenticate. Authentication is allowed without IP address restrictions, if not specified.

  Example:

  ```
  whitelistSourceRanges:
  - 192.168.42.0/24
  ```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **spec.highAvailability**
  boolean

  Enables high availability mode for `dex-authenticator`.

  When this mode is enabled, multiple pod replicas will be deployed to ensure fault tolerance.

  Default: `false`

- **spec.keepUsersLoggedInFor**
  string

  Maximum period of user inactivity during which the user session remains active.

  Set as a string specifying hours, minutes, or seconds: `30m`, `20s`, `2h30m10s`, `24h`.

  Specified value can’t be shorter or equal to the `idTokenTTL` value set in the module settings. If you set a value that is shorter or equal to `idTokenTTL`, the effective logout time will be `idTokenTTL + 1s`.

  To force logout sooner than the default `idTokenTTL`, decrease the `idTokenTTL` parameter value in the module settings as well.

  Default: `168h`

- **spec.nodeSelector**
  object

  If specified, the `dex-authenticator` pods nodeSelector.

  Format: the standard `nodeSelector` list. Instance pods inherit this field as is.

- **spec.resources**
  object

  Container resources (CPU/memory) — requests/limits.

  If this field is set, the specified values are applied and VPA is disabled for this instance. If the field is not set, behavior remains the same: when the VPA module is enabled, resources are managed by VPA, when it is disabled, default values are used.

  Resources can be specified either directly for the `dex-authenticator` container (for backward compatibility) or in nested objects — separately for each container.

  - **spec.resources.limits**
    object

    Upper limit of resources for the `dex-authenticator` container (CPU/memory). This field is kept for backward compatibility.

  - **spec.resources.redis**
    object

    Resources of the `redis` container (CPU/memory) — requests/limits.

    - **spec.resources.redis.limits**
      object

      Upper limit of resources for the `redis` container (CPU/memory).

    - **spec.resources.redis.requests**

object

Guaranteed minimum resources for the `redis` container (CPU/memory).

- **spec.resources.requests**
  object

  Guaranteed minimum resources for the `dex-authenticator` container (CPU/memory). This field is kept for backward compatibility.

- **spec.sendAuthorizationHeader**
  boolean

  Request to application will be sent with `Authorization: Bearer` header when the option is switched to true.

- **spec.tolerations**
  array of objects

  If specified the `dex-authenticator` pods tolerations.

  Format: the standard toleration list. Instance pods inherit this field as is.

  - **spec.tolerations.effect**
    string

    Defines the taint's effect this toleration is associated with (it matches any effect if empty).

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **spec.tolerations.key**
    string

    Defines the taint's key this toleration is associated with (it matches any key if empty).

    If the key is omitted (emplty), then `operator` must be set to `Exists` (which means it matches any `value` and `key`).

  - **spec.tolerations.operator**
    string

    Defines how the key and its value are linked — whether the key must be `Equal` to the value or can have any value.

    Setting it to `Exists` is equivalent to the value having any value so that the pod with the specified toleration can match the corresponding taint.

    Default: `Equal`

    Allowed values: `Exists`, `Equal`

  - **spec.tolerations.tolerationSeconds**
    integer

    Specifies the period (in seconds) that the scheduler waits before evicting a Pod from a node if the toleration no longer matches the taint (**only** applicable to the `NoExecute` effect, otherwise ignored).

    If the parameter **is not set**, the Pod will not be evicted from the node if the toleration no longer matches the taint. If the parameter **is set to zero** (or negative), the Pod will be evicted immediately if the toleration no longer matches the taint.

    The default is: not set.

  - **spec.tolerations.value**
    string

    The value that the toleration must match.

    Must be empty if the `operator` parameter is set to `Exists`.

Deprecated resource. Support for the resource might be removed in a later release.

After the `DexAuthenticator` object appears in the namespace, the following objects will be created:

- Deployment containing OAuth2-Proxy and Redis containers;
- Service, pointing to OAuth2-Proxy;
- Ingress resource, configured to receive requests on `https://<applicationDomain>/dex-`

`authenticator` and send it to a service side;
- HTTPRoute resource (optional), configured to receive requests on `https://<applicationDomain>/dex-authenticator` and send it to a service side;
- Secrets, needed to access Dex.

Warning. DexAuthenticator only works with HTTPS. The application Ingress must have TLS configured.

Warning. After restarting a pod with an OAuth2-Proxy, the current Access Token and ID Token will be queried (using the refresh token) and stored in a Redis memory.

Example:

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  name: app-name
  namespace: app-namespace
spec:
  applicationDomain: app-name.kube.my-domain.com
  sendAuthorizationHeader: false
  applicationIngressCertificateSecretName: ingress-tls
  applicationIngressClassName: nginx
  keepUsersLoggedInFor: 720h
  allowedGroups:
  - everyone
  - admins
  whitelistSourceRanges:
  - 1.1.1.1/32
  - 192.168.0.0/24
  additionalApplications:
  - domain: additional-app-name.kube.my-domain.com
    ingressSecretName: ingress-tls
    ingressClassName: nginx
    signOutURL: "/logout"
    whitelistSourceRanges:
    - 2.2.2.2/32
```

- **spec**
  object

  Required value

  - **spec.additionalApplications**
    array of objects

    A list of additional applications for which user authentication is required.

    - **spec.additionalApplications.domain**
      string

      Required value

      An application domain from which the user request will be redirected for authentication in Dex.

      > **Warning.** Specify a DNS name that complies with RFC 1123 (subdomain) and does not include an HTTP scheme. IP addresses are not allowed.

      Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

    - **spec.additionalApplications.ingressClassName**
      string

      Required value

      The name of the Ingress class to be used in the Ingress resource (it must match the Ingress class name for the application domain).

Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

- **spec.additionalApplications.ingressSecretName**
  string

  The name of the Secret containing the TLS certificate (from the domain) used in the applications Ingress resource. The Secret must be in the same namespace as DexAuthenticator.

  Pattern: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

- **spec.additionalApplications.signOutURL**
  string

  The URL of the application from which requests will be redirected to the sign-out URL of the `dex-authenticator`.

  It is used in the application to direct logout requests. A separate Ingress resource will be created for the specified URL, and requests to it will be redirected to `dex-authenticator`.

  The value must not contain line breaks.

  Pattern: `^[^\r\n]*$`

  Maximum length: `2048`

- **spec.additionalApplications.whitelistSourceRanges**
  array of strings

  CIDRs that are allowed to authenticate. Authentication is allowed without IP address restrictions, If not specified.

  Example:

  ```
  whitelistSourceRanges:
  - 192.168.42.0/24
  ```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

- **spec.allowedEmails**
  array of strings

  A list of emails of users are allowed to connect to the client.

  By default, all users can connect.

  Values must not contain line breaks.

  Default: `All emails are allowed.`

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `320`

- **spec.allowedGroups**
  array of strings

  A list of groups whose members are allowed to connect to the client.

  By default, all groups can connect.

  Values must not contain line breaks.

  Default: `All groups are allowed.`

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

Maximum length: `512`

- **spec.applicationDomain**
  string

  Required value

  An address for the application (DNS domain for Ingress resource) from which the user request will be redirected for authentication in Dex.

  > **Warning.** Specify a DNS name that complies with RFC 1123 (subdomain) and does not include an HTTP scheme. IP addresses are not allowed.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  applicationDomain: my-app.domain.com
  ```

- **spec.applicationIngressCertificateSecretName**
  string

  The name of the Secret containing the TLS certificate for the application domain (is used in the Ingress resource). The Secret must be located in the same namespace as the DexAuthenticator.

  Pattern: `^(|[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)$`

  Example:

  ```
  applicationIngressCertificateSecretName: ingress-tls
  ```

- **spec.applicationIngressClassName**
  string

  Required value

  The name of the Ingress class to be used in the Ingress resource (it must match the Ingress class name for the application domain).

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Example:

  ```
  applicationIngressClassName: nginx
  ```

- **spec.gatewayAPI**
  object

  A set of parameters to use in HTTPRoute object.

  > **Warning.** Only applicable in clusters with Gateway API support.

  > **Warning.** Define the same ListenerSet object which is used to publish the application domain.

  - **spec.gatewayAPI.applicationHTTPRouteListenerSetName**
    string

    Required value

    Name of the ListenerSet object from the Gateway API to be referred to from the HTTPRoute.

    Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

    Maximum length: `253`

Example:

```
applicationHTTPRouteListenerSetName: default-listenerset
```

- **spec.gatewayAPI.applicationHTTPRouteListenerSetNamespace**
  string

  Namespace where the ListenerSet object from the Gateway API is deployed to be referred to from the HTTPRoute.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Maximum length: `63`

  Example:

  ```
  applicationHTTPRouteListenerSetNamespace: default
  ```

- **spec.gatewayAPI.applicationHTTPRouteListenerSetSectionName**
  string

  Section name from the ListenerSet object in the Gateway API the HTTPRoute refers to.

  Pattern: `^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`

  Maximum length: `253`

  Example:

  ```
  applicationHTTPRouteListenerSetSectionName: default-listener
  ```

- **spec.highAvailability**
  boolean

  Enables high availability mode for `dex-authenticator`.

  When this mode is enabled, multiple pod replicas will be deployed to ensure fault tolerance.

  Default: `false`

- **spec.keepUsersLoggedInFor**
  string

  Maximum period of user inactivity during which the user session remains active.

  Set as a string specifying hours, minutes, or seconds: `30m`, `20s`, `2h30m10s`, `24h`.

  Specified value can’t be shorter or equal to the `idTokenTTL` value set in the module settings. If you set a value that is shorter or equal to `idTokenTTL`, the effective logout time will be `idTokenTTL + 1s`.

  To force logout sooner than the default `idTokenTTL`, decrease the `idTokenTTL` parameter value in the module settings as well.

  Default: `168h`

  Example:

  ```
  keepUsersLoggedInFor: 24h
  ```

- **spec.nodeSelector**
  object

  If specified, the `dex-authenticator` pods nodeSelector.

  Format: the standard `nodeSelector` list. Instance pods inherit this field as is.

- **spec.resources**
  object

  Container resources (CPU/memory) — requests/limits.

  If this field is set, the specified values are applied and VPA is disabled for this instance. If the field is not set, behavior remains the same: when the VPA module is enabled, resources are managed by VPA, when it is disabled, default values are used.

Resources can be specified either directly for the `dex-authenticator` container (for backward compatibility) or in nested objects — separately for each container.

- **spec.resources.limits**
  object

  Upper limit of resources for the `dex-authenticator` container (CPU/memory). This field is kept for backward compatibility.

- **spec.resources.redis**
  object

  Resources of the `redis` container (CPU/memory) — requests/limits.

  - **spec.resources.redis.limits**
    object

    Upper limit of resources for the `redis` container (CPU/memory)

  - **spec.resources.redis.requests**
    object

    Guaranteed minimum resources for the `redis` container (CPU/memory).

- **spec.resources.requests**
  object

  Guaranteed minimum resources for the `dex-authenticator` container (CPU/memory). This field is kept for backward compatibility.

- **spec.sendAuthorizationHeader**
  boolean

  Request to application will be sent with `Authorization: Bearer` header when the option is switched to true.

- **spec.signOutURL**
  string

  The URL of the application from which requests will be redirected to the sign-out URL of the `dex-authenticator`.

  It is used in the application to direct logout requests. A separate Ingress resource will be created for the specified URL, and requests to it will be redirected to `dex-authenticator`.

  The value must not contain line breaks.

  Pattern: `^[^\r\n]*$`

  Maximum length: `2048`

- **spec.tolerations**
  array of objects

  If specified the `dex-authenticator` pods tolerations.

  Format: the standard toleration list. Instance pods inherit this field as is.

  - **spec.tolerations.effect**
    string

    Defines the taint’s effect this toleration is associated with (it matches any effect if empty).

    Allowed values: `NoSchedule`, `PreferNoSchedule`, `NoExecute`

  - **spec.tolerations.key**
    string

    Defines the taint’s key this toleration is associated with (it matches any key if empty).

    If the key is omitted (emplty), then `operator` must be set to `Exists` (which means it matches any `value` and `key`).

  - **spec.tolerations.operator**
    string

    Defines how the key and its value are linked — whether the key must be `Equal` to the value or can have any value.

Setting it to `Exists` is equivalent to the value having any value so that the pod with the specified toleration can match the corresponding taint.

Default: `Equal`

Allowed values: `Exists`, `Equal`

- **spec.tolerations.tolerationSeconds**
  integer

  Specifies the period (in seconds) that the scheduler waits before evicting a Pod from a node if the toleration no longer matches the taint (**only** applicable to the `NoExecute` effect, otherwise ignored).

  If the parameter **is not set**, the Pod will not be evicted from the node if the toleration no longer matches the taint. If the parameter **is set to zero** (or negative), the Pod will be evicted immediately if the toleration no longer matches the taint.

  The default is: not set.

- **spec.tolerations.value**
  string

  The value that the toleration must match.

  Must be empty if the `operator` parameter is set to `Exists`.

- **spec.whitelistSourceRanges**
  array of strings

  CIDRs that are allowed to authenticate. Authentication is allowed without IP address restrictions, If not specified.

  Example:

```
whitelistSourceRanges:
- 192.168.42.0/24
```

  - **Element of the array**
    string

    Pattern: `^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/[0-9]{1,2}$`

DexClient

Scope: Namespaced

- v1
- v1alpha1

Allows applications that support DC authentication to interact with Dex.

After the `DexClient` object appears in the cluster:

- Dex will register a client with a `dex-client-<NAME>@<NAMESPACE>` **clientID**, where `<NAME>` and `<NAMESPACE>` are `metadata.name` and `metadata.namespace` of the DexClient object, respectively.
- A `dex-client-<NAME>` Secret containing the client access password (**clientSecret**) will be created in the corresponding namespace (where `<NAME>` is `metadata.name` of the DexClient object).

Usage example...

- **spec**
  object

  Required value

  - **spec.allowedEmails**
    array of strings

    A list of emails of users are allowed to connect to the client.

    By default, all users can connect.

    Values must not contain line breaks.

Default: `All users are allowed.`

- **Element of the array**
  string

  Pattern: `^[^\r\n]*$`

  Maximum length: `320`

- **spec.allowedGroups**
  array of strings

  A list of groups whose members are allowed to connect to the client; **By default**, all groups can connect.

  Values must not contain line breaks.

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `512`

- **spec.redirectURIs**
  array of strings

  Array or urls that Dex can redirect to after successful authentication.

  Values must not contain line breaks.

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `2048`

- **spec.secretMetadata**
  object

  A list of labels and an annotations that will be transferred to the metadata secrets of the DexClient resource.

  - **spec.secretMetadata.annotations**
    object
  - **spec.secretMetadata.labels**
    object

- **spec.trustedPeers**
  array of strings

  OAuth2 client IDs that allowed cross authentication with the current client.

  Values must not contain line breaks.

  Details...

  - **Element of the array**
    string

    Pattern: `^[^\r\n]*$`

    Maximum length: `256`

Deprecated resource. Support for the resource might be removed in a later release.

Allows applications that support DC authentication to interact with Dex.

After the `DexClient` object appears in the cluster:

- Dex will register a client with a `dex-client-<NAME>@<NAMESPACE>` **clientID**, where `<NAME>` and `<NAMESPACE>` are `metadata.name` and `metadata.namespace` of the DexClient object, respectively.
- A `dex-client-<NAME>` Secret containing the client access password (**clientSecret**) will be created in the corresponding namespace (where `<NAME>` is `metadata.name` of the DexClient object).

Usage example...

- **spec**
  object

  Required value

  - **spec.allowedEmails**
    array of strings

    A list of emails of users are allowed to connect to the client.

    By default, all users can connect.

    Values must not contain line breaks.

    Default: `All users are allowed.`

    - **Element of the array**
      string

      Pattern: `^[^\r\n]*$`

      Maximum length: `320`

  - **spec.allowedGroups**
    array of strings

    A list of groups whose members are allowed to connect to the client; **By default**, all groups can connect.

    Values must not contain line breaks.

    - **Element of the array**
      string

      Pattern: `^[^\r\n]*$`

      Maximum length: `512`

  - **spec.redirectURIs**
    array of strings

    Array or urls that Dex can redirect to after successful authentication.

    Values must not contain line breaks.

    - **Element of the array**
      string

      Pattern: `^[^\r\n]*$`

      Maximum length: `2048`

  - **spec.secretMetadata**
    object

    A list of labels and an annotations that will be transferred to the metadata secrets of the DexClient resource.

    - **spec.secretMetadata.annotations**
      object
    - **spec.secretMetadata.labels**
      object
  - **spec.trustedPeers**
    array of strings

    OAuth2 client IDs that allowed cross authentication with the current client.

    Values must not contain line breaks.

    Details…

    - **Element of the array**
      string

      Pattern: `^[^\r\n]*$`

      Maximum length: `256`

DexProvider

Scope: Cluster

- v1
- v1alpha1

Defines the configuration for connecting a third-party provider.

With it, you can flexibly configure the integration of the account directory with Kubernetes.

Usage example…

Warning. It is forbidden to use users and groups with the `system:` prefix. Authentication attempts by such users or members of such groups will be rejected, and a corresponding warning will appear in the `kube-apiserver` logs.

- **spec**
  object

  Required value

  - **spec.bitbucketCloud**
    object

    Parameters of the Bitbucket Cloud (intended for the `type: BitbucketCloud`).

    - **spec.bitbucketCloud.clientID**
      string

      Required value

      Team application ID from BitbucketCloud (Key).

    - **spec.bitbucketCloud.clientSecret**
      string

      Required value

      Team application secret key from BitbucketCloud.

    - **spec.bitbucketCloud.includeTeamGroups**
      boolean

      Optional parameter to include team groups.

      If enabled, the groups claim of Dex id_token will looks like this:

      ```
      ["my_team", "my_team/administrators", "my_team/members"]
      ```

      Default: `false`

    - **spec.bitbucketCloud.teams**
      array of strings

      A list of allowed Bitbucket Cloud teams (filter).

      The user token will contain a set intersection of Bitbucket Cloud teams and teams from this list. If the set is empty, the authorization will be considered unsuccessful.

      The user token will contain the user teams in the `groups` claim (similar to other providers).

  - **spec.crowd**
    object

    Parameters of the Crowd (intended for the `type: Crowd`).

    - **spec.crowd.baseURL**
      string

      Required value

      Base part of Attlassian Crowd URL.

      Example:

      ```
      baseURL: https://crowd.example.com/crowd
      ```

        - **spec.crowd.clientID**
          string

          Required value

          Application ID from Atlassian Crowd (Application Name).

        - **spec.crowd.clientSecret**
          string

          Required value

          Application secret key from Atlassian Crowd (Password).

        - **spec.crowd.enableBasicAuth**
          boolean

          Enables basic authorization for the Kubernetes API server.

          The username and password of the user from the application created in Crowd are used as credentials for basic authorization (you can enable it only if there is just one provider of the OIDC/Crowd type). Works **only** if the `publishAPI` is enabled.

          Authorization and group data obtained from an IdP are stored in the cache for 10 seconds.

        - **spec.crowd.groups**
          array of strings

          A list of allowed Crowd groups (filter).

          The user token will contain a set intersection of Crowd groups and groups from this list. If the set is empty, the authorization will be considered unsuccessful.

          The user token will contain all Crowd groups if the parameter is not set.

        - **spec.crowd.usernamePrompt**
          string

          Prompt for username field.

          Default: `Crowd username`

    - **spec.displayName**
      string

      Required value

      The provider name to show on the authentication provider selection page. The selection page will not be displayed if there is only one provider configured.

    - **spec.enabled**
      boolean

      When false, this provider is ignored by Dex config renderer.

      Default: `true`

    - **spec.github**
      object

      Parameters of the GitHub provider (intended for the `type: Github` case only).

        - **spec.github.clientID**
          string

          Required value

          Organization application ID from GitHub.

        - **spec.github.clientSecret**
          string

          Required value

          Organization application secret key from GitHub.

        - **spec.github.orgs**
          array of objects

Filter for user organizations. ID token will contain only organizations from this list. If the user is not in any organization from this list, an authorization will fail.

By default, all organizations allowed.

- **spec.github.orgs.name**
  string

  Required value

  Name of organization.

- **spec.github.orgs.teams**
  array of strings

  A list of allowed GitHub teams (filter).

  The user token will contain a set intersection of teams from GitHub and teams from this list. If the set is empty, the authorization will be considered unsuccessful.

  The user token will contain all GitHub teams if the parameter is not set.

- **spec.github.teamNameField**
  string

  As an example, group claims for member of ‘Site Reliability Engineers’ in Acme organization would yield:

  - [‘acme:Site Reliability Engineers’] for ‘Name’
  - [‘acme:site-reliability-engineers’] for ‘Slug’
  - [‘acme:Site Reliability Engineers’, ‘acme:site-reliability-engineers’] for ‘Both’

  ‘name’ will be used by default.

  Default: `Name`

  Allowed values: `Name`, `Slug`, `Both`

- **spec.github.useLoginAsID**
  boolean

  Flag which will switch from using the internal GitHub id to the users handle (@mention) as the user id. It is possible for a user to change their own user name but it is very rare for them to do so.

  Equals to false by default.

- **spec.gitlab**
  object

  Parameters of the GitLab provider (intended for the `type: Gitlab` case only).

  - **spec.gitlab.baseURL**
    string

    Base part of GitLab URL.

    Example:

    ```
    baseURL: https://gitlab.example.com
    ```

  - **spec.gitlab.clientID**
    string

    Required value

    Application ID from GitLab.

  - **spec.gitlab.clientSecret**
    string

    Required value

    Application secret key from GitLab.

  - **spec.gitlab.groups**
    array of strings

A list (filter) of allowed GitLab groups (**group paths** and not names).

The user token will contain a set intersection of GitLab groups and groups from this list. If the set is empty, the authorization will be considered unsuccessful.

The user token will contain all GitLab groups if the parameter is not set;

- **spec.gitlab.rootCAData**
  string

  A CA chain to validate the provider in PEM format.

  Example:

  ```
  rootCAData: |
    -----BEGIN CERTIFICATE-----
    MIIFaDC...
    -----END CERTIFICATE-----
  ```

- **spec.gitlab.useLoginAsID**
  boolean

  Flag to switch from using the internal GitLab id to the users handle (@mention) as the user id. It is possible for a user to change their own user name but it is very rare for them to do so.

  Equals to false by default.

- **spec.ldap**
  object

  Parameters of the LDAP.

  - **spec.ldap.bindDN**
    string

    The DN for an application service account. The connector uses these credentials to search for users and groups. Not required if the LDAP server provides access for anonymous auth.

    Example:

    ```
    bindDN: uid=serviceaccount,cn=users,dc=example,dc=com
    ```

  - **spec.ldap.bindPW**
    string

    Password for read-only service account. Please note that if the bind password contains a `$`, it has to be saved in an environment variable which should be given as the value to `bindPW`.

    Example:

    ```
    bindPW: password
    ```

  - **spec.ldap.enableBasicAuth**
    boolean

    Enables Basic Authentication for access to the Kubernetes API server.

    The credentials are the Crowd user’s username and password (from the application configured for this provider).

    Limitations:

    - works only when `publishAPI` is enabled;
    - can be enabled for only one OIDC or Crowd provider in the cluster.

    Authentication data and group membership obtained from the IdP are cached for 10 seconds.

  - **spec.ldap.groupSearch**
    object

Group search queries for groups given a user entry. Details

- **spec.ldap.groupSearch.baseDN**
  string

  Required value

  BaseDN to start the search from.

  Example:

  ```
  baseDN: cn=users,dc=example,dc=com
  ```

- **spec.ldap.groupSearch.filter**
  string

  Optional filter to apply when searching the directory.

  Example:

  ```
  filter: "(objectClass=person)"
  ```

- **spec.ldap.groupSearch.nameAttr**
  string

  Required value

  Represents group name.

  Example:

  ```
  nameAttr: name
  ```

- **spec.ldap.groupSearch.userMatchers**
  array of objects

  Required value

  Following list contains field pairs that are used to match a user to a group. It adds a requirement to the filter that an attribute in the group must match the user’s attribute value.

  - **spec.ldap.groupSearch.userMatchers.groupAttr**
    string

    Required value

    The name of the attribute that stores the group member names.

    Example:

    ```
    groupAttr: member
    ```

  - **spec.ldap.groupSearch.userMatchers.userAttr**
    string

    Required value

    The name of the attribute that stores the user name.

    Example:

    ```
    userAttr: uid
    ```

- **spec.ldap.host**
  string

  Required value

  Host and optional port of the LDAP server in the form “host:port”. If the port is not supplied, it will be guessed based on “insecureNoSSL”, and “startTLS” flags. 389 for insecure or StartTLS connections, 636 otherwise.

  Example:

```
host: ldap.example.com:636
```

- **spec.ldap.insecureNoSSL**
  boolean

  Following field is required if the LDAP host is not using TLS (port 389). This option inherently leaks passwords to anyone on the same network as Dex. Equals to false by default.

  Default: `false`

- **spec.ldap.insecureSkipVerify**
  boolean

  If a custom certificate isn’t provided, this option can be used to turn off TLS certificate checks. As noted, it is insecure and shouldn’t be used outside of explorative phases.

  Default: `false`

- **spec.ldap.kerberos**
  object

  Optional Kerberos (SPNEGO) SSO settings for LDAP.

  - **spec.ldap.kerberos.enabled**
    boolean

    Enable Kerberos SPNEGO flow for this LDAP provider.

    Default: `false`

  - **spec.ldap.kerberos.expectedRealm**
    string

    Optional realm to require (case-insensitive).

  - **spec.ldap.kerberos.fallbackToPassword**
    boolean

    If `true`, in the absence of/incorrect `Authorization: Negotiate` header, a login/password form will be displayed instead of a 401 response with `WWW-Authenticate: Negotiate`.

    Default: `false`

  - **spec.ldap.kerberos.keytabSecretName**
    string

    Name of Secret containing key ‘krb5.keytab’ to mount for Dex.

  - **spec.ldap.kerberos.krb5ConfConfigMapName**
    string

    Optional ConfigMap name containing key ‘krb5.conf’ to mount at /etc/krb5.conf.

  - **spec.ldap.kerberos.krb5ConfSecretName**
    string

    Optional Secret name containing key ‘krb5.conf’ to mount at /etc/krb5.conf.

  - **spec.ldap.kerberos.usernameFromPrincipal**
    string

    How to map Kerberos principal to LDAP username.

    Default: `Localpart`

    Allowed values: `Localpart`, `SAMAccountName`, `UserPrincipalName`

- **spec.ldap.rootCAData**
  string

  A CA chain to validate the provider in PEM format.

  Example:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

- **spec.ldap.startTLS**
  boolean

  When connecting to the server, connect using the ldap:// protocol then issue a StartTLS command. If unspecified, connections will use the ldaps:// protocol

  Default: `false`

- **spec.ldap.userSearch**
  object

  Required value

  User search maps a username and password entered by a user to a LDAP entry.
  Details...

  - **spec.ldap.userSearch.baseDN**
    string

    Required value

    BaseDN to start the search from.

    Example:

    ```
    baseDN: cn=users,dc=example,dc=com
    ```

  - **spec.ldap.userSearch.emailAttr**
    string

    Required value

    LDAP attribute that will be matched to Dex user email entry. When an email address is not available, use another value unique to the user, like uid.

    Example:

    ```
    emailAttr: mail
    ```

  - **spec.ldap.userSearch.filter**
    string

    Optional filter to apply when searching the directory.

    Example:

    ```
    filter: "(objectClass=person)"
    ```

  - **spec.ldap.userSearch.idAttr**
    string

    Required value

    LDAP attribute that will be matched to Dex user id entry.

    Example:

    ```
    idAttr: uid
    ```

  - **spec.ldap.userSearch.nameAttr**
    string

    LDAP attribute that will be matched to Dex user name entry. No default value provided.

    Example:

```
nameAttr: name
```

- **spec.ldap.userSearch.username**
  string

  Required value

  Username attribute used for comparing user entries. This will be translated and combined with the other filter as “(=)".

  Example:

  ```
  username: uid
  ```

- **spec.ldap.usernamePrompt**
  string

  The attribute to display in the provided password prompt. If unset, will display “LDAP Username”.

  Default: `LDAP username`

  Example:

  ```
  usernamePrompt: SSO Username
  ```

- **spec.oidc**
  object

  Parameters of the OIDC (intended for the `type: OIDC`).

  - **spec.oidc.allowedGroups**
    array of strings

    A list of groups the user must belong to in order to authenticate successfully.

    If none of the specified groups match the user’s groups, authentication will be denied.

    If the parameter is omitted, no group-based filtering will be applied.

  - **spec.oidc.basicAuthUnsupported**
    boolean

    Use POST requests to interact with the provider instead of including the token in the Basic Authorization header. Generally, Dex automatically determines the type of request to make, while in some cases enabling this parameter can help.

    Default: `false`

  - **spec.oidc.claimMapping**
    object

    Some providers return non-standard claims (eg. mail). Claim mappings are hints for Dex how to map claims to standard OIDC claims.

    Dex can only map a non-standard claim to a standard one if it’s not included in the id_token returned by OIDC provider.

    - **spec.oidc.claimMapping.email**
      string

      The claim to use as the user email.

      Default: `email`

    - **spec.oidc.claimMapping.groups**
      string

      The claim to use as the user groups.

      Default: `groups`

    - **spec.oidc.claimMapping.preferred_username**
      string

      The claim to use as the user preferred username.

Default: `preferred_username`

- **spec.oidc.claimMappingOverride**
  boolean

  If enabled, the claim mapping will override the standard OIDC claims.

  By default, the claim mapping will be used only if the standard OIDC claims are not present, e.g., if there is no `email` claim in the id_token, the `claimMapping.email` will be used.

  Default: `false`

- **spec.oidc.clientID**
  string

  Required value

  OIDC issuer application ID.

- **spec.oidc.clientSecret**
  string

  Required value

  OIDC issuer application secret key.

- **spec.oidc.enableBasicAuth**
  boolean

  Enables basic authorization for the Kubernetes API server.

  The username and password of the user from the application created in OIDC are used as credentials for basic authorization (you can enable it only if there is just one provider of the OIDC/Crowd type). Works **only** if the publishAPI parameter is enabled.

  Authorization and group data obtained from an IdP are stored in the cache for 10 seconds.

- **spec.oidc.getUserInfo**
  boolean

  Request additional info about the authenticated user.

  Learn more here...

  Default: `false`

- **spec.oidc.insecureSkipEmailVerified**
  boolean

  Allow authentication for clients without verified email address.

  Default: `false`

- **spec.oidc.insecureSkipVerify**
  boolean

  If a custom certificate isn’t provided, this option can be used to turn off TLS certificate checks. As noted, it is insecure and shouldn’t be used outside of explorative phases.

  Default: `false`

- **spec.oidc.issuer**
  string

  Required value

  Canonical URL of the provider, also used for configuration discovery. This value MUST match the value returned in the provider config discovery.

  Example:

  ```
  issuer: https://accounts.google.com
  ```

- **spec.oidc.promptType**
  string

  Determines if the Issuer should ask for confirmation and provide hints during the

Determines if the Issuer should ask for confirmation and provide hints during the authentication process.

By default, the confirmation will be requested on the first authentication. Possible values may vary depending on the Issuer.

Default: `consent`

- **spec.oidc.rootCAData**
  string

  A CA chain to validate the provider in PEM format.

  Example:

  ```
  rootCAData: |
    -----BEGIN CERTIFICATE-----
    MIIFaDC...
    -----END CERTIFICATE-----
  ```

- **spec.oidc.scopes**
  array of strings

  List of additional scopes to request in token response.

  A scope is a single token and must not contain whitespace: the OAuth2 protocol passes scopes as a space-delimited list.

  Default: `["openid","profile","email","groups","offline_access"]`

  - **Element of the array**
    string

    Pattern: `^\S+$`

- **spec.oidc.userIDKey**
  string

  The claim to use as the user id.

  Default: `sub`

- **spec.oidc.userNameKey**
  string

  The claim to use as the user name.

  Default: `name`

- **spec.saml**
  object

  Parameters of a SAML provider (intended for the `type: SAML`).

  - **spec.saml.allowedGroups**
    array of strings

    If set, only members of at least one of the listed groups will be allowed to log in.

  - **spec.saml.emailAttr**
    string

    SAML attribute to map to the email claim.

    Default: `email`

  - **spec.saml.entityIssuer**
    string

    Issuer value expected in the SAML response.

    Defaults to the callback URL.

  - **spec.saml.filterGroups**
    boolean

    If `true`, the connector will filter the groups claim to only include groups from the `allowedGroups` list.

Default: `false`

- **spec.saml.groupsAttr**
  string

  SAML attribute to map to the groups claim.

- **spec.saml.groupsDelim**
  string

  Delimiter to split the groups attribute value if it is a single string.

- **spec.saml.insecureSkipSignatureValidation**
  boolean

  Skip signature validation of SAML responses. This is insecure and should only be used for testing.

  Default: `false`

- **spec.saml.nameIDPolicyFormat**
  string

  The NameIDPolicy format to request from the IdP.

  Allowed values: `persistent`, `emailAddress`, `unspecified`, `X509SubjectName`, `WindowsDomainQualifiedName`, `transient`

- **spec.saml.redirectURI**
  string

  Callback URI.

  Overrides the default callback URL. Should be set to `https://<dex-domain>/callback`.

- **spec.saml.rootCAData**
  string

  A CA chain to validate the provider in PEM format.

  Example:

  ```
  rootCAData: |
    -----BEGIN CERTIFICATE-----
    MIIFaDC...
    -----END CERTIFICATE-----
  ```

- **spec.saml.ssoIssuer**
  string

  Issuer value expected in the SAML response from the IdP.

- **spec.saml.ssoURL**
  string

  Required value

  SSO URL used for POST value.

  Example:

  ```
  ssoURL: https://idp.example.com/saml/sso
  ```

- **spec.saml.usernameAttr**
  string

  SAML attribute to map to the username claim.

  Default: `name`

- **spec.type**
  string

  Required value

Type of authentication provider.

Allowed values: `Github`, `Gitlab`, `BitbucketCloud`, `Crowd`, `OIDC`, `LDAP`, `SAML`

Deprecated resource. Support for the resource might be removed in a later release.

Defines the configuration for connecting a third-party provider.

With it, you can flexibly configure the integration of the account directory with Kubernetes.

Usage example...

- **spec**
  object

  Required value

  - **spec.bitbucketCloud**
    object

    Parameters of the Bitbucket Cloud (intended for the `type: BitbucketCloud`).

    - **spec.bitbucketCloud.clientID**
      string

      Required value

      Team application ID from BitbucketCloud (Key).

    - **spec.bitbucketCloud.clientSecret**
      string

      Required value

      Team application secret key from BitbucketCloud.

    - **spec.bitbucketCloud.includeTeamGroups**
      boolean

      Optional parameter to include team groups.

      If enabled, the groups claim of Dex id_token will looks like this:

      ```
      ["my_team", "my_team/administrators", "my_team/members"]
      ```

      Default: `false`

    - **spec.bitbucketCloud.teams**
      array of strings

      A list of allowed Bitbucket Cloud teams (filter).

      The user token will contain a set intersection of Bitbucket Cloud teams and teams from this list. If the set is empty, the authorization will be considered unsuccessful.

      The user token will contain the user teams in the `groups` claim (similar to other providers).

  - **spec.crowd**
    object

    Parameters of the Crowd (intended for the `type: Crowd`).

    - **spec.crowd.baseURL**
      string

      Required value

      Base part of Attlassian Crowd URL.

      Example:

      ```
      baseURL: https://crowd.example.com/crowd
      ```

    - **spec.crowd.clientID**
      string

Required value

Application ID from Atlassian Crowd (Application Name).

- **spec.crowd.clientSecret**
  string

  Required value

  Application secret key from Atlassian Crowd (Password).

- **spec.crowd.enableBasicAuth**
  boolean

  Enables basic authorization for the Kubernetes API server.

  The username and password of the user from the application created in Crowd are used as credentials for basic authorization (you can enable it only if there is just one provider of the OIDC/Crowd type). Works **only** if the `publishAPI` is enabled.

  Authorization and group data obtained from an IdP are stored in the cache for 10 seconds.

- **spec.crowd.groups**
  array of strings

  A list of allowed Crowd groups (filter).

  The user token will contain a set intersection of Crowd groups and groups from this list. If the set is empty, the authorization will be considered unsuccessful.

  The user token will contain all Crowd groups if the parameter is not set.

- **spec.crowd.usernamePrompt**
  string

  Prompt for username field.

  Default: `Crowd username`

- **spec.displayName**
  string

  Required value

  The provider name to show on the authentication provider selection page. The selection page will not be displayed if there is only one provider configured.

- **spec.enabled**
  boolean

  When false, this provider is ignored by Dex config renderer.

  Default: `true`

- **spec.github**
  object

  Parameters of the GitHub provider (intended for the `type: Github` case only).

  - **spec.github.clientID**
    string

    Required value

    Organization application ID from GitHub.

  - **spec.github.clientSecret**
    string

    Required value

    Organization application secret key from GitHub.

  - **spec.github.orgs**
    array of objects

    Filter for user organizations. ID token will contain only organizations from this list. If the user is not in any organization from this list, an authorization will fail.

By default, all organizations allowed.

- **spec.github.orgs.name**
  string

  Required value

  Name of organization.

- **spec.github.orgs.teams**
  array of strings

  A list of allowed GitHub teams (filter).

  The user token will contain a set intersection of teams from GitHub and teams from this list. If the set is empty, the authorization will be considered unsuccessful.

  The user token will contain all GitHub teams if the parameter is not set.

- **spec.github.teamNameField**
  string

  As an example, group claims for member of ‘Site Reliability Engineers’ in Acme organization would yield:

  - [‘acme:Site Reliability Engineers’] for ‘name’
  - [‘acme:site-reliability-engineers’] for ‘slug’
  - [‘acme:Site Reliability Engineers’, ‘acme:site-reliability-engineers’] for ‘both’

  ‘name’ will be used by default.

  Default: `name`

  Allowed values: `name`, `slug`, `both`

- **spec.github.useLoginAsID**
  boolean

  Flag which will switch from using the internal GitHub id to the users handle (@mention) as the user id. It is possible for a user to change their own user name but it is very rare for them to do so.

  Equals to false by default.

- **spec.gitlab**
  object

  Parameters of the GitLab provider (intended for the `type: Gitlab` case only).

  - **spec.gitlab.baseURL**
    string

    Base part of GitLab URL.

    Example:

    ```
    baseURL: https://gitlab.example.com
    ```

  - **spec.gitlab.clientID**
    string

    Required value

    Application ID from GitLab.

  - **spec.gitlab.clientSecret**
    string

    Required value

    Application secret key from GitLab.

  - **spec.gitlab.groups**
    array of strings

    A list (filter) of allowed GitLab groups (**group paths** and not names).

    The user token will contain a set intersection of GitLab groups and groups from this list.

If the set is empty, the authorization will be considered unsuccessful.

The user token will contain all GitLab groups if the parameter is not set;

- **spec.gitlab.rootCAData**
  string

  A CA chain to validate the provider in PEM format.

  Example:

  ```
  rootCAData: |
    -----BEGIN CERTIFICATE-----
    MIIFaDC...
    -----END CERTIFICATE-----
  ```

- **spec.gitlab.useLoginAsID**
  boolean

  Flag to switch from using the internal GitLab id to the users handle (@mention) as the user id. It is possible for a user to change their own user name but it is very rare for them to do so.

  Equals to false by default.

- **spec.ldap**
  object

  Parameters of the LDAP.

  - **spec.ldap.bindDN**
    string

    The DN for an application service account. The connector uses these credentials to search for users and groups. Not required if the LDAP server provides access for anonymous auth.

    Example:

    ```
    bindDN: uid=serviceaccount,cn=users,dc=example,dc=com
    ```

  - **spec.ldap.bindPW**
    string

    Password for read-only service account. Please note that if the bind password contains a `$`, it has to be saved in an environment variable which should be given as the value to `bindPW`.

    Example:

    ```
    bindPW: password
    ```

  - **spec.ldap.enableBasicAuth**
    boolean

    Enables Basic Authentication for access to the Kubernetes API server.

    The credentials are the Crowd user’s username and password (from the application configured for this provider).

    Limitations:

    - works only when `publishAPI` is enabled;
    - can be enabled for only one OIDC or Crowd provider in the cluster.

    Authentication data and group membership obtained from the IdP are cached for 10 seconds.

  - **spec.ldap.groupSearch**
    object

    Group search queries for groups given a user entry. Details

    - **spec.ldap.groupSearch.baseDN**
      string

Required value

BaseDN to start the search from.

Example:

```
baseDN: cn=users,dc=example,dc=com
```

- **spec.ldap.groupSearch.filter**
  string

  Optional filter to apply when searching the directory.

  Example:

  ```
  filter: "(objectClass=person)"
  ```

- **spec.ldap.groupSearch.nameAttr**
  string

  Required value

  Represents group name.

  Example:

  ```
  nameAttr: name
  ```

- **spec.ldap.groupSearch.userMatchers**
  array of objects

  Required value

  Following list contains field pairs that are used to match a user to a group. It adds a requirement to the filter that an attribute in the group must match the user’s attribute value.

  - **spec.ldap.groupSearch.userMatchers.groupAttr**
    string

    Required value

    The name of the attribute that stores the group member names.

    Example:

    ```
    groupAttr: member
    ```

  - **spec.ldap.groupSearch.userMatchers.userAttr**
    string

    Required value

    The name of the attribute that stores the user name.

    Example:

    ```
    userAttr: uid
    ```

- **spec.ldap.host**
  string

  Required value

  Host and optional port of the LDAP server in the form “host:port”. If the port is not supplied, it will be guessed based on “insecureNoSSL”, and “startTLS” flags. 389 for insecure or StartTLS connections, 636 otherwise.

  Example:

  ```
  host: ldap.example.com:636
  ```

- **spec.ldap.insecureNoSSL**

boolean

Following field is required if the LDAP host is not using TLS (port 389). This option inherently leaks passwords to anyone on the same network as Dex. Equals to false by default.

Default: `false`

- **spec.ldap.insecureSkipVerify**
  boolean

  If a custom certificate isn’t provided, this option can be used to turn off TLS certificate checks. As noted, it is insecure and shouldn’t be used outside of explorative phases.

  Default: `false`

- **spec.ldap.kerberos**
  object

  Optional Kerberos (SPNEGO) SSO settings for LDAP.

  - **spec.ldap.kerberos.enabled**
    boolean

    Enable Kerberos SPNEGO flow for this LDAP provider.

    Default: `false`

  - **spec.ldap.kerberos.expectedRealm**
    string

    Optional realm to require (case-insensitive).

  - **spec.ldap.kerberos.fallbackToPassword**
    boolean

    If `true`, in the absence of/incorrect `Authorization: Negotiate` header, a login/password form will be displayed instead of a 401 response with `WWW-Authenticate: Negotiate`.

    Default: `false`

  - **spec.ldap.kerberos.keytabSecretName**
    string

    Name of Secret containing key ‘krb5.keytab’ to mount for Dex.

  - **spec.ldap.kerberos.krb5ConfConfigMapName**
    string

    Optional ConfigMap name containing key ‘krb5.conf’ to mount at /etc/krb5.conf.

  - **spec.ldap.kerberos.krb5ConfSecretName**
    string

    Optional Secret name containing key ‘krb5.conf’ to mount at /etc/krb5.conf.

  - **spec.ldap.kerberos.usernameFromPrincipal**
    string

    How to map Kerberos principal to LDAP username.

    Default: `Localpart`

    Allowed values: `Localpart`, `SAMAccountName`, `UserPrincipalName`

- **spec.ldap.rootCAData**
  string

  A CA chain to validate the provider in PEM format.

  Example:

```
rootCAData: |
  -----BEGIN CERTIFICATE-----
  MIIFaDC...
  -----END CERTIFICATE-----
```

- **spec.ldap.startTLS**
  boolean

  When connecting to the server, connect using the ldap:// protocol then issue a StartTLS command. If unspecified, connections will use the ldaps:// protocol

  Default: `false`

- **spec.ldap.userSearch**
  object

  Required value

  User search maps a username and password entered by a user to a LDAP entry.
  Details...

  - **spec.ldap.userSearch.baseDN**
    string

    Required value

    BaseDN to start the search from.

    Example:

    ```
    baseDN: cn=users,dc=example,dc=com
    ```

  - **spec.ldap.userSearch.emailAttr**
    string

    Required value

    LDAP attribute that will be matched to Dex user email entry. When an email address is not available, use another value unique to the user, like uid.

    Example:

    ```
    emailAttr: mail
    ```

  - **spec.ldap.userSearch.filter**
    string

    Optional filter to apply when searching the directory.

    Example:

    ```
    filter: "(objectClass=person)"
    ```

  - **spec.ldap.userSearch.idAttr**
    string

    Required value

    LDAP attribute that will be matched to Dex user id entry.

    Example:

    ```
    idAttr: uid
    ```

  - **spec.ldap.userSearch.nameAttr**
    string

    LDAP attribute that will be matched to Dex user name entry. No default value provided.

    Example:

    ```
    nameAttr: name
    ```

  - **spec.ldap.userSearch.username**
    string

    Required value

Username attribute used for comparing user entries. This will be translated and combined with the other filter as “(=)".

Example:

```
username: uid
```

- **spec.ldap.usernamePrompt**
  string

  The attribute to display in the provided password prompt. If unset, will display “LDAP Username”.

  Default: `LDAP username`

  Example:

  ```
  usernamePrompt: SSO Username
  ```

- **spec.oidc**
  object

  Parameters of the OIDC (intended for the `type: OIDC`).

  - **spec.oidc.allowedGroups**
    array of strings

    A list of groups the user must belong to in order to authenticate successfully.

    If none of the specified groups match the user’s groups, authentication will be denied.

    If the parameter is omitted, no group-based filtering will be applied.

  - **spec.oidc.basicAuthUnsupported**
    boolean

    Use POST requests to interact with the provider instead of including the token in the Basic Authorization header. Generally, Dex automatically determines the type of request to make, while in some cases enabling this parameter can help.

    Default: `false`

  - **spec.oidc.claimMapping**
    object

    Some providers return non-standard claims (eg. mail). Claim mappings are hints for Dex how to map claims to standard OIDC claims.

    Dex can only map a non-standard claim to a standard one if it’s not included in the id_token returned by OIDC provider.

    - **spec.oidc.claimMapping.email**
      string

      The claim to use as the user email.

      Default: `email`

    - **spec.oidc.claimMapping.groups**
      string

      The claim to use as the user groups.

      Default: `groups`

    - **spec.oidc.claimMapping.preferred_username**
      string

      The claim to use as the user preferred username.

      Default: `preferred_username`

  - **spec.oidc.claimMappingOverride**
    boolean

    If enabled, the claim mapping will override the standard OIDC claims.

By default, the claim mapping will be used only if the standard OIDC claims are not present, e.g., if there is no `email` claim in the id_token, the `claimMapping.email` will be used.

Default: `false`

- **spec.oidc.clientID**
  string

  Required value

  OIDC issuer application ID.

- **spec.oidc.clientSecret**
  string

  Required value

  OIDC issuer application secret key.

- **spec.oidc.enableBasicAuth**
  boolean

  Enables basic authorization for the Kubernetes API server.

  The username and password of the user from the application created in OIDC are used as credentials for basic authorization (you can enable it only if there is just one provider of the OIDC/Crowd type). Works **only** if the publishAPI parameter is enabled.

  Authorization and group data obtained from an IdP are stored in the cache for 10 seconds.

- **spec.oidc.getUserInfo**
  boolean

  Request additional info about the authenticated user.

  Learn more here...

  Default: `false`

- **spec.oidc.insecureSkipEmailVerified**
  boolean

  Allow authentication for clients without verified email address.

  Default: `false`

- **spec.oidc.insecureSkipVerify**
  boolean

  If a custom certificate isn’t provided, this option can be used to turn off TLS certificate checks. As noted, it is insecure and shouldn’t be used outside of explorative phases.

  Default: `false`

- **spec.oidc.issuer**
  string

  Required value

  Canonical URL of the provider, also used for configuration discovery. This value MUST match the value returned in the provider config discovery.

  Example:

  ```
  issuer: https://accounts.google.com
  ```

- **spec.oidc.promptType**
  string

  Determines if the Issuer should ask for confirmation and provide hints during the authentication process.

  By default, the confirmation will be requested on the first authentication. Possible values may vary depending on the Issuer.

  Default: `consent`

- **spec.oidc.rootCAData**
  string

  A CA chain to validate the provider in PEM format.

  Example:

  ```
  rootCAData: |
    -----BEGIN CERTIFICATE-----
    MIIFaDC...
    -----END CERTIFICATE-----
  ```

- **spec.oidc.scopes**
  array of strings

  List of additional scopes to request in token response.

  A scope is a single token and must not contain whitespace: the OAuth2 protocol passes scopes as a space-delimited list.

  Default: `["openid","profile","email","groups","offline_access"]`

  - **Element of the array**
    string

    Pattern: `^\S+$`

- **spec.oidc.userIDKey**
  string

  The claim to use as the user id.

  Default: `sub`

- **spec.oidc.userNameKey**
  string

  The claim to use as the user name.

  Default: `name`

- **spec.saml**
  object

  Parameters of a SAML provider (intended for the `type: SAML`).

  - **spec.saml.allowedGroups**
    array of strings

    If set, only members of at least one of the listed groups will be allowed to log in.

  - **spec.saml.emailAttr**
    string

    SAML attribute to map to the email claim.

    Default: `email`

  - **spec.saml.entityIssuer**
    string

    Issuer value expected in the SAML response.

    Defaults to the callback URL.

  - **spec.saml.filterGroups**
    boolean

    If `true`, the connector will filter the groups claim to only include groups from the `allowedGroups` list.

    Default: `false`

  - **spec.saml.groupsAttr**
    string

    SAML attribute to map to the groups claim.

- **spec.saml.groupsDelim**
  string

  Delimiter to split the groups attribute value if it is a single string.

- **spec.saml.insecureSkipSignatureValidation**
  boolean

  Skip signature validation of SAML responses. This is insecure and should only be used for testing.

  Default: `false`

- **spec.saml.nameIDPolicyFormat**
  string

  The NameIDPolicy format to request from the IdP.

  Allowed values: `persistent`, `emailAddress`, `unspecified`, `X509SubjectName`, `WindowsDomainQualifiedName`, `transient`

- **spec.saml.redirectURI**
  string

  Callback URI.

  Overrides the default callback URL. Should be set to `https://<dex-domain>/callback`.

- **spec.saml.rootCAData**
  string

  A CA chain to validate the provider in PEM format.

  Example:

  ```
  rootCAData: |
    -----BEGIN CERTIFICATE-----
    MIIFaDC...
    -----END CERTIFICATE-----
  ```

- **spec.saml.ssoIssuer**
  string

  Issuer value expected in the SAML response from the IdP.

- **spec.saml.ssoURL**
  string

  Required value

  SSO URL used for POST value.

  Example:

  ```
  ssoURL: https://idp.example.com/saml/sso
  ```

- **spec.saml.usernameAttr**
  string

  SAML attribute to map to the username claim.

  Default: `name`

- **spec.type**
  string

  Required value

  Type of authentication provider.

  Allowed values: `Github`, `Gitlab`, `BitbucketCloud`, `Crowd`, `OIDC`, `LDAP`, `SAML`

Group

Scope: Cluster
Version: v1alpha1

Contains information about a user group.

- **spec**
  object

  Required value

  - **spec.members**
    array of objects

    Required value

    List of group members.

    - **spec.members.kind**
      string

      Required value

      Type of a group member.

      Allowed values: `User`, `Group`

    - **spec.members.name**
      string

      Required value

      User or group name.

  - **spec.name**
    string

    Required value

    Unique group name.

  - **spec.status**
    object
    - **spec.status.errors**
      array of objects
      - **spec.status.errors.message**
        string
      - **spec.status.errors.objectRef**
        object
        - **spec.status.errors.objectRef.kind**
          string

          Allowed values: `User`, `Group`

        - **spec.status.errors.objectRef.name**
          string

UserOperation

Scope: Cluster

- v1
- v1alpha1

Describes an operation to be applied to a user (password reset, 2FA reset, lock/unlock).

- **spec**
  object

  Required value

  - **spec.initiatorType**
    string

    Indicates who initiated the operation. Possible values:

    - `admin`: By an administrator.
    - `system`: Automatically by the system.
    - `self`: By the user.

- **spec.lock**
  object

  Parameters for locking the user.

  - **spec.lock.for**
    string

    Lock duration. Should be specified in the following format: `30s`, `10m`, `1h`, `2h30m`.

- **spec.resetPassword**
  object

  Parameters for password reset.

  - **spec.resetPassword.newPasswordHash**
    string

    Bcrypt hash of the new password.

- **spec.type**
  string

  Required value

  Type of the user operation. Possible values:

  - `ResetPassword`: Password reset.
  - `Reset2FA`: Two-factor authentication reset.
  - `Lock`: User lock.
  - `Unlock`: User unlock.
- **spec.user**
  string

  Required value

  Name (username) of the target User resource.

  Example:

  ```
  user: jane.doe
  ```

- **status**
  object
  - **status.completedAt**
    string

    Time when the operation was completed.

  - **status.message**
    string

    Optional message about the result or failure of the operation.

  - **status.phase**
    string

    Current phase of the operation. Possible values:

    - `Succeeded`: Operation was completed successfully.
    - `Failed`: Operation failed.

    Allowed values: `Succeeded`, `Failed`

Deprecated resource. Support for the resource might be removed in a later release.

Describes an operation to be applied to a user (password reset, 2FA reset, lock/unlock).

- **spec**
  object

  Required value

  - **spec.initiatorType**
    string

    Indicates who initiated the operation. Possible values:

        - `admin`: By an administrator.
        - `system`: Automatically by the system.
        - `self`: By the user.
    - **spec.lock**
      object

      Parameters for locking the user.

        - **spec.lock.for**
          string

          Lock duration. Should be specified in the following format: `30s`, `10m`, `1h`, `2h30m`.
    - **spec.resetPassword**
      object

      Parameters for password reset.

        - **spec.resetPassword.newPasswordHash**
          string

          Bcrypt hash of the new password.
    - **spec.type**
      string

      Required value

      Type of the user operation. Possible values:

        - `ResetPassword`: Password reset.
        - `Reset2FA`: Two-factor authentication reset.
        - `Lock`: User lock.
        - `Unlock`: User unlock.
    - **spec.user**
      string

      Required value

      Name (username) of the target User resource.

      Example:

      ```
      user: jane.doe
      ```

- **status**
  object
    - **status.completedAt**
      string

      Time when the operation was completed.
    - **status.message**
      string

      Optional message about the result or failure of the operation.
    - **status.phase**
      string

      Current phase of the operation. Possible values:

        - `Succeeded`: Operation completed successfully.
        - `Failed`: Operation failed.

      Allowed values: `Succeeded`, `Failed`

User

Scope: Cluster

- v1
- v1alpha1

Contains information about the static user.

Usage example...

- **spec**
  object

  Required value

  - **spec.email**
    string

    Required value

    User email. Must be in lowercase.

    Caution! Note that if used together with the user-authz module, you must specify an `email` to grant rights to the specific user as the user name in the ClusterAuthorizationRule CR.

    Important! Email addresses are case-insensitive for uniqueness validation. Dex automatically converts emails to lowercase during authentication. For new users, uppercase letters in email addresses are not allowed. Existing users with uppercase emails can be updated without changing the email field.

    Example:

    ```
    email: user@domain.com
    ```

  - **spec.groups**
    Deprecated
    array of strings

    Static user groups.

    Since the parameter has been deprecated, use the Group resource to add users to groups.

  - **spec.password**
    string

    Required value

    User password hash in plaintext or Base64 encoded.

    Use the following command to encode the password hash in Base64: `echo -n '<PASSWORD>' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0; echo`. Alternatively, you can use an online service (such as https://bcrypt-generator.com/).

    Example:

    ```
    password:
    JDJ5JDEwJGRNWGVGUVBkdUdYYVMyWDFPcGdZdk9HSy81LkdsNm5sdU9mUkhnNWlQdDhuSlh6Szh
    peS5H
    ```

  - **spec.ttl**
    string

    Static user TTL.

    It is specified as a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

    You can only set the TTL once. The `expireAt` date will not be updated if you change it again.

    Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

    Example:

    ```
    ttl: 24h
    ```

  - **spec.userID**
    Deprecated
    string

    Unique issuer user ID. It equals to .metadata.name.

    Deprecated and shouldn’t be set manually.

- **status**
  object
  - **status.expireAt**
    string

    User account expiration date.

    - It is shown only of the `.spec.ttl` field is set.
    - The user account will be deleted at the specified date.
    - This parameter is synchronized every 5 minutes. There may be a time lag between the moment specified in this field and the moment of actual deletion of the user account.
  - **status.groups**
    array of strings

    Static user groups.

  - **status.lock**
    object

    Information about user account lock. If present, the user cannot authenticate until the specified time or until the lock is manually lifted by an administrator.

    Administrative actions on the account (lock, unlock, forced password reset, 2FA reset) are performed via the UserOperation resource. Self-service password reset by the user is also performed via UserOperation (`initiatorType: self`). When an administrator locks the account or initiates a forced password reset, all of the user's active sessions (Dex `OfflineSessions` and `RefreshToken` objects) are deleted, so the user is forced to re-authenticate; for a forced password reset they are presented with a password change form on the next successful login.

    - **status.lock.message**
      string

      Human-readable explanation of the lock reason.

    - **status.lock.reason**
      string

      Machine-readable reason for the account lock. Used for programmatic checks and policy enforcement.

      Allowed values: `PasswordPolicyLockout`, `LockedByAdministrator`

    - **status.lock.state**
      boolean

      Indicates whether the user account is currently locked.

      - `true` — account is locked.
      - `false` — account is active.
    - **status.lock.until**
      string

      The time until which the user is locked, in RFC3339 format. After this time, the account lock will be automatically lifted.

Deprecated resource. Support for the resource might be removed in a later release.

Contains information about the static user.

Usage example...

- **spec**
  object

  Required value

  - **spec.email**
    string

    Required value

    User email. Must be in lowercase.

    Caution! Note that if used together with the user-authz module, you must specify an `email` to grant rights to the specific user as the user name in the ClusterAuthorizationRule CR.

    Important! Email addresses are case-insensitive for uniqueness validation. Dex

automatically converts emails to lowercase during authentication. For new users, uppercase letters in email addresses are not allowed. Existing users with uppercase emails can be updated without changing the email field.

Example:

```
email: user@domain.com
```

- **spec.groups**
  Deprecated
  array of strings

  Static user groups.

  Since the parameter has been deprecated, use the Group resource to add users to groups.

- **spec.password**
  string

  Required value

  User password hash in plaintext or Base64 encoded.

  Use the following command to encode the password hash in Base64: `echo -n '<PASSWORD>' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0; echo`. Alternatively, you can use an online service (such as https://bcrypt-generator.com/).

  > **i** **Caution.** If a password policy with complexity checking higher than `None` is used, the user will have to change a password set this way upon their first successful login.

  Example:

  ```
  password:
  JDJ5JDEwJGRNWGVGUVBkdUdYYVMyWDFPcGdZdk9HSy81LkdsNm5sdU9mUkhnNWlQdDhuSlh6SzhpeS5H
  ```

- **spec.ttl**
  string

  Static user TTL.

  It is specified as a string containing the time unit in hours and minutes: 30m, 1h, 2h30m, 24h.

  You can only set the TTL once. The `expireAt` date will not be updated if you change it again.

  Pattern: `^([0-9]+h([0-9]+m)?|[0-9]+m)$`

  Example:

  ```
  ttl: 24h
  ```

- **spec.userID**
  string

  Unique issuer user ID. It equals to .metadata.name by default.

  Example:

  ```
  userID: '08a8684b-db88-4b73-90a9-3cd1661f5466'
  ```

- **status**
  object
  - **status.expireAt**
    string

    User account expiration date.

    - It is shown only of the `.spec.ttl` field is set.
    - The user account will be deleted at the specified date.

        - This parameter is synchronized every 5 minutes. There may be a time lag between the moment specified in this field and the moment of actual deletion of the user account.
    - **status.groups**
      array of strings

      Static user groups.

    - **status.lock**
      object

      Information about user account lock. If present with `state: true`, the user cannot authenticate until the specified time or until the lock is manually lifted by an administrator.

      Administrative actions on the account (lock, unlock, forced password reset, 2FA reset) are performed via the UserOperation resource. Self-service password reset by the user is also performed via UserOperation (`initiatorType: self`). When an administrator locks the account or initiates a forced password reset, all of the user's active sessions (Dex `OfflineSessions` and `RefreshToken` objects) are deleted, so the user is forced to re-authenticate; for a forced password reset they are presented with a password change form on the next successful login.

        - **status.lock.message**
          string

          Human-readable explanation of the lock reason.

        - **status.lock.reason**
          string

          Machine-readable reason for the account lock. Used for programmatic checks and policy enforcement.

          Allowed values: `PasswordPolicyLockout`, `LockedByAdministrator`

        - **status.lock.state**
          boolean

          Indicates whether the user account is currently locked.

            - `true` — account is locked.
            - `false` — account is active.
        - **status.lock.until**
          string

          The time until which the user is locked, in RFC3339 format. After this time, the account lock will be automatically lifted.

Usage

> **i** The module lifecycle stage: General Availability

An example of the module configuration

The example shows the configuration of the `user-authn` module in the Deckhouse Kubernetes Platform.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    kubeconfigGenerator:
    - id: direct
      masterURI: https://159.89.5.247:6443
      description: "Direct access to kubernetes API"
    publishAPI:
      enabled: true
```

Configuring a provider

Checking provider connectivity

The web UI provider details page has a **Check connection** action. It creates a DexProviderCheck resource and waits for the hook to write the result to its status.

The check verifies that:

- the referenced DexProvider exists and is enabled;
- the Dex discovery endpoint inside the cluster is reachable;
- the external provider endpoint is reachable.

For OIDC providers, the check also reads the discovery document and JWKS. For LDAP providers, it verifies TCP/TLS/StartTLS reachability and, when Kerberos is enabled, that the keytab Secret exists. The check does not perform a full login flow and does not validate a test user password.

GitHub

The example shows the provider’s settings for integration with GitHub.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: github
spec:
  type: Github
  displayName: My Company Github
  # Optional: disable the provider without deleting CR
  # enabled: false
  github:
    clientID: plainstring
    clientSecret: plainstring
```

In your GitHub organization, create a new application:

To do this, go to `Settings` -> `Developer settings` -> `OAuth Aps` -> `Register a new OAuth application` and specify the `https://dex.<modules.publicDomainTemplate>/callback` address as the `Authorization callback URL`.

Paste the generated `Client ID` and `Client Secret` into the DexProvider custom resource.

If the GitHub organization is managed by the client, go to `Settings` -> `Applications` -> `Authorized OAuth Apps` -> `<name of created OAuth App>` and request confirmation by clicking on `Send Request`. Then ask the client to confirm the request that will be sent to him by email.

GitLab

The example shows the provider’s settings for integration with GitLab.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: gitlab
spec:
  type: Gitlab
  displayName: Dedicated GitLab
  gitlab:
    baseURL: https://gitlab.example.com
    clientID: plainstring
    clientSecret: plainstring
    groups:
    - administrators
    - users
```

Create a new application in the GitLab project.

To do this, you need to:

- **self-hosted**: go to `Admin area` -> `Application` -> `New application` and specify the `https://dex.<modules.publicDomainTemplate>/callback` address as the `Redirect URI (Callback URL)` and set scopes `read_user`, `openid`;
- **cloud GitLab.com**: under the main project account, go to `User Settings` -> `Application` -> `New application` and specify the `https://dex.<modules.publicDomainTemplate>/callback` address as the `Redirect URI (Callback URL)`; also, don't forget to set scopes `read_user`, `openid`;
- (for GitLab version starting with 16) enable the `Trusted`/`Trusted applications are automatically authorized on GitLab OAuth flow` checkbox when creating an application.

Paste the generated `Application ID` and `Secret` into the DexProvider custom resource.

Atlassian Crowd

The example shows the provider's settings for integration with Atlassian Crowd.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: crowd
spec:
  type: Crowd
  displayName: Crowd
  crowd:
    baseURL: https://crowd.example.com/crowd
    clientID: plainstring
    clientSecret: plainstring
    enableBasicAuth: true
    groups:
    - administrators
    - users
```

Create a new `Generic` application in the corresponding Atlassian Crowd project.

To do this, go to `Applications` -> `Add application`.

Paste the generated `Application Name` and `Password` into the DexProvider custom resource.

CROWD groups are specified in the lowercase format for the custom resource `DexProvider`.

Bitbucket Cloud

The example shows the provider's settings for integration with Bitbucket Cloud.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: bitbucket
spec:
  type: BitbucketCloud
  displayName: Bitbucket
  bitbucketCloud:
    clientID: plainstring
    clientSecret: plainstring
    includeTeamGroups: true
    teams:
    - administrators
    - users
```

Create a new OAuth consumer in the Bitbucket's team menu.

To do this, go to `Settings` -> `OAuth consumers` -> `New application` and specify the `https://dex.<modules.publicDomainTemplate>/callback` address as the `Callback URL`. Also, allow access for `Account: Read` and `Workspace membership: Read`.

Paste the generated `Key` and `Secret` into the DexProvider custom resource.

OIDC (OpenID Connect)

Authentication through the OIDC provider requires registering a client (or "creating an application"). Please refer to the provider's documentation on how to do it (e.g., Okta, Keycloak, Gluu).

Paste the generated `clientID` and `clientSecret` into the DexProvider custom resource.

Keycloak

After selecting a `realm` to configure, adding a user in the Users section, and creating a client in the Clients section with authentication enabled, which is required to generate the `clientSecret`, you need to perform the following steps:

- Create a `scope` named `groups` in the Client scopes section and assign it the predefined mapping `groups`. ("Client scopes" → "Client scope details" → "Mappers" → "Add predefined mappers")
- In the previously created client, add this `scope` in the Client scopes tab ("Clients" → "Client details" → "Client Scopes" → "Add client scope").
- In the "Valid redirect URIs", "Valid post logout redirect URIs", and "Web origins" fields of the client configuration, specify `https://dex.<publicDomainTemplate>/*`, where `publicDomainTemplate` is a value of the parameter in the `global` module config.

The example shows the provider's settings for integration with Keycloak.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: keycloak
spec:
  type: OIDC
  displayName: My Company Keycloak
  oidc:
    issuer: https://keycloak.my-company.com/realms/myrealm # Use the name of your
realm
    clientID: plainstring
    clientSecret: plainstring
    insecureSkipEmailVerified: true
    getUserInfo: true
    scopes:
      - openid
      - profile
      - email
      - groups
```

If email verification is not enabled in Keycloak, to properly use it as an identity provider, adjust the `Client Scopes` settings in one of the following ways:

- Delete the `Email verified` mapping (“Client Scopes” → “Email” → “Mappers”). This is required for proper processing of the `insecureSkipEmailVerified` field when it’s set to `true` and for correct permission assignment to users with unverified emails.
- If you can’t modify or delete the `Email verified` mapping, create a new Client Scope named `email_dkp` (or any other name) and add two mappings:
  - `email`: “Client Scopes” → `email_dkp` → “Add mapper” → “From predefined mappers” → `email`.
  - `email verified`: “Client Scopes” → `email_dkp` → “Add mapper” → “By configuration” → “Hardcoded claim”. Specify the following fields:
    - “Name”: `email verified`
    - “Token Claim Name”: `emailVerified`
    - “Claim value”: `true`
    - “Claim JSON Type”: `boolean`

  After that, in the client registered for the DKP cluster in “Clients”, change `Client scopes` from `email` to `email_dkp`.

  In the DexProvider resource, specify `insecureSkipEmailVerified: true` and in the `.spec.oidc.scopes` field, change the Client Scope name to `email_dkp` following the example:

  ```
  scopes:
    - openid
    - profile
    - email_dkp
    - groups
  ```

Okta

The example shows the provider’s settings for integration with Okta.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: okta
spec:
  type: OIDC
  displayName: My Company Okta
  oidc:
    issuer: https://my-company.okta.com
    clientID: plainstring
    clientSecret: plainstring
    insecureSkipEmailVerified: true
    getUserInfo: true
```

Blitz Identity Provider

Note that you must specify a URL to redirect the user after authorization when registering the application with the Blitz Identity Provider. When using `DexProvider`, you must specify `https://dex.<publicDomainTemplate>/`, where `publicDomainTemplate` is the cluster's DNS name template as defined in the `global` module.

The example below shows the provider settings for integration with Blitz Identity Provider.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: blitz
spec:
  displayName: Blitz Identity Provider
  oidc:
    basicAuthUnsupported: false
    claimMapping:
      email: email
      groups: your_claim # Claim for getting user groups, configured on the Blitz
    clientID: clientID
    clientSecret: clientSecret
    getUserInfo: true
    insecureSkipEmailVerified: true # Set to true if there is no need to verify the
user's email
    insecureSkipVerify: false
    issuer: https://yourdomain.idblitz.ru/blitz
    promptType: consent
    scopes:
    - profile
    - openid
    userIDKey: sub
    userNameKey: email
  type: OIDC
```

For the application logout to work correctly (the token being revoked so that re-authorization is required), set `login` as the value of the 'promptType` parameter.

To ensure granular user access to applications, you have to:

- Add the `allowedUserGroups` parameter to the `ModuleConfig` of the target application.
- Add the user to the groups (group names should be the same for Blitz and Deckhouse).

Below is an example for prometheus:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: prometheus
spec:
  version: 2
  settings:
    auth:
      allowedUserGroups:
        - adm-grafana-access
        - grafana-access
```

LDAP

The example shows the provider's settings for integration with Active Directory.

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: active-directory
spec:
  type: LDAP
  displayName: Active Directory
  ldap:
    host: ad.example.com:636
    insecureSkipVerify: true

    bindDN: cn=Administrator,cn=users,dc=example,dc=com
    bindPW: admin0!

    usernamePrompt: Email Address

    enableBasicAuth: true

    userSearch:
      baseDN: cn=Users,dc=example,dc=com
      filter: "(objectClass=person)"
      username: userPrincipalName
      idAttr: DN
      emailAttr: userPrincipalName
      nameAttr: cn

    groupSearch:
      baseDN: cn=Users,dc=example,dc=com
      filter: "(objectClass=group)"
      userMatchers:
      - userAttr: DN
        groupAttr: member
      nameAttr: cn
```

Configuring Basic Authentication

To enable Basic Authentication for the Kubernetes API using LDAP credentials:

1. Ensure that the `publishAPI` parameter is enabled in the `user-authn` module configuration.
2. Set `enableBasicAuth: true` in your LDAP DexProvider resource.

> **Warning**. Only one provider in the cluster can have `enableBasicAuth` enabled.

After configuration, users can access the Kubernetes API via `kubectl`, using their LDAP username and

password.

Example `kubeconfig` for the user:

```
apiVersion: v1
kind: Config
clusters:
- name: my-cluster
  cluster:
    server: https://api.example.com
    # Path to CA certificate or insecure-skip-tls-verify: true
    certificate-authority: /path/to/ca.crt
users:
- name: ldap-user
  user:
    username: janedoe@example.com
    password: userpassword
contexts:
- name: default
  context:
    cluster: my-cluster
    user: ldap-user
current-context: default
```

Kerberos (SPNEGO) SSO for LDAP

Dex supports passwordless Kerberos (SPNEGO) flow for the LDAP connector. When enabled, a browser that trusts the Dex host will send `Authorization: Negotiate …` and Dex will validate the Kerberos ticket using a service keytab, skip the login form, map the Kerberos principal to an LDAP username, query groups, and complete the OIDC flow.

Minimal example extending the LDAP provider spec:

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: active-directory
spec:
  type: LDAP
  displayName: Active Directory
  ldap:
    host: ad.example.com:636
    bindDN: cn=Administrator,cn=users,dc=example,dc=com
    bindPW: admin0!
    userSearch:
      baseDN: cn=Users,dc=example,dc=com
      username: sAMAccountName
      idAttr: uid
      emailAttr: mail
      nameAttr: cn
    groupSearch:
      baseDN: cn=Users,dc=example,dc=com
      nameAttr: cn
      userMatchers:
      - userAttr: uid
        groupAttr: memberUid
    kerberos:
      enabled: true
      keytabSecretName: dex-kerberos-keytab  # Secret in d8-user-authn with key
'krb5.keytab'
      expectedRealm: EXAMPLE.COM           # optional, case-insensitive match
      usernameFromPrincipal: sAMAccountName  #
localpart|sAMAccountName|userPrincipalName
      fallbackToPassword: false            # default false; if true, render form when
header missing/invalid
```

Notes:

- The Secret `dex-kerberos-keytab` must exist in the `d8-user-authn` namespace and have a data key named exactly `krb5.keytab`.
- A single Dex Pod can serve multiple LDAP+Kerberos providers. Each provider mounts its own keytab; a shared `krb5.conf` is not required (Dex validates tickets offline using the keytab). To configure authentication, create a read-only user (service account) in LDAP.

Specify the generated user path and password in the `bindDN` and `bindPW` fields of the DexProvider custom resource. Enter the password into the `bindPW` in plain-text format (unencrypted). Strategies involving the passing of hashed passwords are not supported.

You can omit these settings if anonymous read access is configured for LDAP.

SAML

The example shows the provider’s settings for integration with a SAML 2.0 Identity Provider (e.g., AD FS, Okta, Keycloak).

```
apiVersion: deckhouse.io/v1
kind: DexProvider
metadata:
  name: saml-provider
spec:
  type: SAML
  displayName: Corporate SAML
  saml:
    ssoURL: https://saml-idp.example.com/saml/sso
    rootCAData: |
      -----BEGIN CERTIFICATE-----
      MIIFaDC...
      -----END CERTIFICATE-----
    entityIssuer: https://dex.example.com/callback
    ssoIssuer: https://saml-idp.example.com
    usernameAttr: name
    emailAttr: email
    groupsAttr: groups
    nameIDPolicyFormat: persistent
```

To configure the SAML Identity Provider:

1. Register Dex as a Service Provider (SP) with your identity provider using the following settings:
   - **ACS URL (Assertion Consumer Service)**: `https://dex.<modules.publicDomainTemplate>/callback`
   - **Entity ID**: `https://dex.<modules.publicDomainTemplate>/callback`
   - **Name identifier format**: `persistent` or `emailAddress`
2. Configure attribute mapping in the identity provider to send `email`, `name` (username), and `groups` attributes in the SAML assertion.
3. Export the identity provider signing certificate and specify it in the `rootCAData` field of the DexProvider resource.

> **i** SAML does not natively support refresh tokens. Dex caches the user identity from the initial SAML assertion and returns it on subsequent refresh requests. The session lifetime is controlled by the `expiry.refreshTokens` settings in the `user-authn` module configuration.

Configuring the OAuth2 client in Dex for connecting an application

This configuration is suitable for applications that can independently perform oauth2 authentication without using an oauth2 proxy. The `DexClient` custom resource enables applications to use dex.

```
apiVersion: deckhouse.io/v1
kind: DexClient
metadata:
  name: myname
  namespace: mynamespace
spec:
  redirectURIs:
  - https://app.example.com/callback
  - https://app.example.com/callback-reserve
  allowedGroups:
  - Everyone
  - admins
  trustedPeers:
  - opendistro-sibling
```

After the `DexClient` custom resource is created, Dex will register a client with a `dex-client-myname@mynamespace` ID (**clientID**).

The client access password (**clientSecret**) will be stored in the secret object:

```
apiVersion: v1
kind: Secret
metadata:
  name: dex-client-myname
  namespace: mynamespace
type: Opaque
data:
  clientSecret: c2VjcmV0
```

Granting an application access to the Kubernetes API

A DexClient or a DexAuthenticator can be allowed to obtain tokens that the Kubernetes API server accepts. To do that, set one of the following annotations on the resource to `"true"`:

- `dexclient.deckhouse.io/allow-access-to-kubernetes` on a DexClient.
- `dexauthenticator.deckhouse.io/allow-access-to-kubernetes` on a DexAuthenticator.

The following is a DexClient configuration example:

```
apiVersion: deckhouse.io/v1
kind: DexClient
metadata:
  name: myname
  namespace: mynamespace
  annotations:
    dexclient.deckhouse.io/allow-access-to-kubernetes: "true"
spec:
  redirectURIs:
  - https://app.example.com/callback
```

The annotation registers the client as a trusted peer of the privileged `kubernetes` OAuth2 client, allowing it to request ID tokens intended for the API server. In such a token, the username is determined by the `email` claim and groups by the `groups` claim. As a result, the application accesses the API server on behalf of the authenticated user and with the permissions granted to that user.

⚠ Access granted this way applies at the cluster level, even though DexClient and DexAuthenticator are namespaced resources. Therefore, only a subject with permissions to modify the `user-authn` module configuration can add the annotation or change its value to `"true"`. For example, this permission is granted by the `d8:manage:permission:module:user-authn:edit` role. Permissions to create DexClient or DexAuthenticator resources in an individual namespace are not sufficient.

Adding the annotation is restricted regardless of its value, including `"false"`. This is required for compatibility with DKP versions earlier than 1.78, where access is granted based on the presence of the annotation regardless of its value. If access to the Kubernetes API server is not required, do not add the annotation.

The restriction does not apply to resources that already have the annotation. A user with permissions to modify such a resource can continue to modify it, including removing the annotation or changing its value to disable access.

Local Authentication

Local authentication provides user verification and access management with support for configurable password policies, two-factor authentication (2FA), and group management.
The implementation complies with OWASP recommendations, ensuring reliable protection of access to the cluster and applications without requiring integration with external authentication systems.

Creating a user

Create a password and enter its hash encoded in base64 in the `password` field. The email address must be in lowercase.

Use the command below to calculate the password hash:

```
echo -n '3xAmpl3Pa$$wo#d' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' | base64 -w0;
echo
```

> **i** If the `htpasswd` command is not available, install the appropriate package:
>
> - `apache2-utils` — for Debian-based distributions.
> - `httpd-tools` — for CentOS-based distributions.
> - `apache2-htpasswd` — for ALT Linux.

Alternatively, you can use the online service to calculate the password hash.

Note that in the below example the `ttl` parameter is set.

```
apiVersion: deckhouse.io/v1
kind: User
metadata:
  name: admin
spec:
  email: admin@yourcompany.com
  # echo -n '3xAmpl3Pa$$wo#d' | htpasswd -BinC 10 "" | cut -d: -f2 | tr -d '\n' |
base64 -w0; echo
  password:
'JDJ5JDEwJGRNWGVGUVBkdUdYYVMyWDFPcGdZdk9HSy81LkdsNm5sdU9mUkhnNWlQdDhuSlh6SzhpeS5H'
  ttl: 24h
```

Authorization rules grant permissions to users based on the email address from the issued token. Therefore, a User resource cannot be created if its `spec.email` value matches a `User` subject in an existing AuthorizationRule or ClusterAuthorizationRule resource. This prevents permissions from being unintentionally granted to a new user.

If the match is intentional, for example, if the authorization rule was created in advance, use a different email address or add the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation to the User resource.

Email addresses are converted to lowercase when matched, as this is how they are stored in the token. For example, `Admin@Example.com` matches the `admin@example.com` subject.

The restriction does not apply to existing users whose email addresses match subjects in authorization rules. Such users continue to work, and a warning is issued when a match is detected.

Consider this restriction when managing users and authorization rules declaratively. If a user and the corresponding rule are created at the same time, the rule may be created first, causing creation of the User resource to be rejected. To allow such a match, add the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation to the User manifest.

Deleting a user does not automatically remove the corresponding subject from the authorization rule. As long as the subject remains in the rule, permissions continue to be granted to the specified email address, and a warning is issued that no corresponding user exists. If a user with the same email address is created later, that user receives these permissions. If this behavior is not required, remove the corresponding subject from the authorization rule.

Local user operations

Password reset, 2FA reset, and lock/unlock operations are performed via the UserOperation resource. The `initiatorType` field indicates who initiated the operation: an administrator (`admin`), the system (`system`), or the user (`self`).

Administrative operations

Use the `d8 iam user` commands for administrative actions on local users. They create a UserOperation resource with `initiatorType: admin`, wait for the operation to complete, and print the result.

The `ResetPassword`, `Reset2FA`, and `Lock` operations delete the user's Dex OfflineSessions and RefreshToken objects. This terminates the user's active offline sessions and requires re-authentication.

Interactive password reset example:

```
d8 iam user reset-password admin
```

Example of reading the new password from stdin:

```
echo "N3wPa$$wo#d" | d8 iam user reset-password admin --password-stdin
```

Example of generating a new password automatically:

```
d8 iam user reset-password admin --generate-password
```

If the password is already hashed, pass the bcrypt hash without Base64 encoding:

```
d8 iam user reset-password admin --password-hash '$2y$10$abcdef...'
```

2FA reset example:

```
d8 iam user reset2fa admin
```

Example of locking a user for 30 minutes:

```
d8 iam user lock admin 30m
```

User unlock example:

```
d8 iam user unlock admin
```

By default, commands wait for the operation to complete. To only create a UserOperation and print its name, use the `--wait=false` flag.

Self-service password reset

A local user can reset their own password in the DKP authentication interface. This creates a UserOperation resource with `type: ResetPassword` and `initiatorType: self`.

Self-service password reset is available only for local accounts (the built-in `Local` connector). Users who sign in through external authentication providers must contact the administrator of the corresponding system.

When a user resets their password, the new password must comply with the password policy, and the user’s active sessions are terminated — re-authentication is required.

Adding a user to a group

Users can be grouped to manage access rights. Example manifest of the Group resource for a group:

```
apiVersion: deckhouse.io/v1alpha1
kind: Group
metadata:
  name: admins
spec:
  name: admins
  members:
    - kind: User
      name: admin
```

Where `members` is a list of users belonging to the group.

The group name is stored in the issued token without modification. It is indistinguishable from a group name received from an external authentication provider. Therefore, a Group resource cannot be created if its `spec.name` value matches a `Group` subject in an existing AuthorizationRule or ClusterAuthorizationRule resource. This prevents permissions from being unintentionally granted to

members of a new group.

If the match is intentional, use a different group name or add the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation to the Group resource.

Unlike email addresses, group names are not converted to lowercase when matched. Therefore, group names that differ only in letter case are considered different.

The restriction does not apply to existing groups whose names match subjects in authorization rules. Such groups continue to work, and a warning is issued when a match is detected.

Consider this restriction when managing groups and authorization rules declaratively. If a group and the corresponding rule are created at the same time, the rule may be created first, causing creation of the Group resource to be rejected. To allow such a match, add the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation to the Group manifest.

Deleting a group does not automatically remove the corresponding subject from the authorization rule. As long as the subject remains in the rule, permissions continue to be granted to a group with the specified name. If a group with the same name is created later, its members receive these permissions. If this behavior is not required, remove the corresponding subject from the authorization rule.

Password policy

Password policy settings allow controlling password complexity, rotation, and user lockout:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    passwordPolicy:
      complexityLevel: Fair
      passwordHistoryLimit: 10
      lockout:
        lockDuration: 15m
        maxAttempts: 3
      rotation:
        interval: "30d"
```

Field description:

- `complexityLevel`: Password complexity level.
- `passwordHistoryLimit`: Number of previous passwords stored in the system to prevent their reuse.
- `lockout`: Lockout settings after exceeding the limit of failed login attempts:
  - `lockout.maxAttempts`: Limit of allowed failed login attempts.
  - `lockout.lockDuration`: User lockout duration.
- `rotation`: Password rotation settings:
  - `rotation.interval`: Period for mandatory password change.

Two-factor authentication (2FA)

2FA increases security by requiring a code from a `TOTP` authenticator application (for example, Google Authenticator) during login.

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authn
spec:
  version: 2
  enabled: true
  settings:
    staticUsers2FA:
      enabled: true
      issuerName: "awesome-app"
```

Field description:

- `enabled`: Enables or disables 2FA for all static users.
- `issuerName`: Name displayed in the authenticator application when adding an account.

> **i** After enabling 2FA, each user must register in the authenticator application during their first login.

Assigning permissions to a user or group

For permission configuration, parameters of the custom resource ClusterAuthorizationRule are used.

FAQ

> **i** The module lifecycle stage: General Availability

How to secure my application?

To enable Dex authentication for your application, follow these steps:

1. Create a DexAuthenticator custom resource.

   When you create a DexAuthenticator in a cluster, an instance of oauth2-proxy is created and connected to Dex. The Deployment, Service, Ingress, and Secret objects will be created in the specified namespace.

   Example of the DexAuthenticator custom resource:

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  # Dex authenticator pod name prefix.
  # For example, if the name prefix is `app-name`, then Dex authenticator pods
will look like `app-name-dex-authenticator-7f698684c8-c5cjg`.
  name: app-name
  # Namespace to deploy Dex authenticator to.
  namespace: app-ns
spec:
  # Your application's domain. Requests to it will be redirected for Dex
authentication.
  applicationDomain: "app-name.kube.my-domain.com"
  # A parameter that determines whether to send the `Authorization: Bearer`
header to the application.
  # This one is useful in combination with auth_request in NGINX.
  sendAuthorizationHeader: false
  # The name of the Secret containing the SSL certificate.
  applicationIngressCertificateSecretName: "ingress-tls"
  # The name of the Ingress class to use in the Ingress resource created for the
Dex authenticator.
  applicationIngressClassName: "nginx"
  # The duration of the active user session.
  keepUsersLoggedInFor: "720h"
  # The list of groups whose users are allowed to authenticate.
  allowedGroups:
  - everyone
  - admins
  # The list of addresses and networks for which authentication is allowed.
  whitelistSourceRanges:
  - 1.1.1.1/32
  - 192.168.0.0/24
```

2. Connect your application to Dex.

   For this, add the following annotations to the application's Ingress resource:

   - `nginx.ingress.kubernetes.io/auth-signin: https://$host/dex-authenticator/sign_in`
   - `nginx.ingress.kubernetes.io/auth-response-headers: X-Auth-Request-User,X-Auth-Request-Email`
   - `nginx.ingress.kubernetes.io/auth-url: https://<SERVICE_NAME>.<NS>.svc.{{ C_DOMAIN }}/dex-authenticator/auth`, where:
     - `SERVICE_NAME`: Name of the authenticator's Service. Usually, it is `<NAME>-dex-authenticator` (`<NAME>` is the `metadata.name` of the DexAuthenticator).
     - `NS`: Value of the `metadata.namespace` parameter of the DexAuthenticator.
     - `C_DOMAIN`: Cluster domain (the clusterDomain parameter of the ClusterConfiguration resource).

> **Note:** If the DexAuthenticator `<NAME>` is too long, the Service name may be truncated. To find the correct service name, use the following command (specify the namespace name and DexAuthenticator name):
>
> ```
> d8 k get service -n <NS> -l "deckhouse.io/dex-authenticator-for=<NAME>" -o
> jsonpath='{.items[0].metadata.name}'
> ```

Example of annotations for an application's Ingress resource for connecting to Dex:

```
annotations:
  nginx.ingress.kubernetes.io/auth-signin: https://$host/dex-
authenticator/sign_in
  nginx.ingress.kubernetes.io/auth-url: https://app-name-dex-authenticator.app-
ns.svc.cluster.local/dex-authenticator/auth
  nginx.ingress.kubernetes.io/auth-response-headers: X-Auth-Request-User,X-Auth-
Request-Email
```

⚠ The application Ingress must have TLS configured. DexAuthenticator does not support HTTP-only Ingress resources.

Setting up CIDR-based restrictions

DexAuthenticator does not have a built-in system for managing authentication based on user IP address. Instead, you can use Ingress resource annotations:

- To restrict access by IP and keep Dex authentication, add the annotation with a comma-separated list of allowed CIDRs:

  ```
  nginx.ingress.kubernetes.io/whitelist-source-range: 192.168.0.0/32,1.1.1.1
  ```

- To allow access without Dex authentication for users from specified networks while requiring authentication for others, add the annotation:

  ```
  nginx.ingress.kubernetes.io/satisfy: "any"
  ```

Authentication flow with DexAuthenticator

⚠ DexAuthenticator only works with HTTPS. It does not support Ingress resources configured for HTTP only.

Authentication cookies are set with the `Secure` attribute, which means they are only sent over encrypted HTTPS connections.

Make sure your application Ingress has TLS configured before integrating with DexAuthenticator.

1. Dex redirects the user to the provider's login page in most cases and waits for the user to be redirected back to the `/callback` URL. However, some providers like LDAP or Atlassian Crowd do not support this flow. The user must enter credentials in the Dex login form instead, and Dex will validate them by making a request to the provider's API.
2. DexAuthenticator sets the cookie with the full refresh token (instead of issuing a ticket as for the ID token) because Redis does not persist data. If no ID token is found in Redis by the ticket, the user can request a new ID token by providing the refresh token from the cookie.
3. DexAuthenticator sets the `Authorization` HTTP header to the ID token value from Redis. This is not required for services like `upmeter`, as `upmeter` permissions are less granular. For the Kubernetes Dashboard, it is critical: the Dashboard passes the ID token on to access the Kubernetes API.

How to generate a kubeconfig and access Kubernetes API?

`kubeconfig` for remote access to the cluster via `kubectl` can be generated in the `kubeconfigurator` web interface.

Configure the `publishAPI` parameter:

- Open the `user-authn` module settings (create the ModuleConfig `user-authn` resource if there is none):

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

- Add the following section to the `settings` block and save:

```
publishAPI:
  enabled: true
```

The name `kubeconfig` is reserved for the kubeconfig generation web interface. The URL depends on the `publicDomainTemplate` parameter (for example, for the template that looks like `%s.kube.my`, the kubeconfig generation web interface will be available at `kubeconfig.kube.my`, and for `%s-kube.company.my` — at `kubeconfig-kube.company.my`).

Configuring kube-apiserver

Using the `control-plane-manager` module, DKP automatically configures `kube-apiserver` with the following flags so that the `dashboard` and `kubeconfig-generator` modules can work in the cluster.

kube-apiserver arguments that will be configured

- `--oidc-client-id=kubernetes`
- `--oidc-groups-claim=groups`
- `--oidc-issuer-url=https://dex.%addonsPublicDomainTemplate%/`
- `--oidc-username-claim=email`

When self-signed certificates are used for Dex, one more argument is added and the CA file is mounted in the `apiserver` pod:

- `--oidc-ca-file=/etc/kubernetes/oidc-ca.crt`

The flow of accessing Kubernetes API with generated kubeconfig

1. Before `kube-apiserver` starts, it must request the OIDC provider’s configuration endpoint (Dex in our case) to get the issuer and JWKS endpoint settings.
2. Kubeconfig generator stores the ID token and refresh token in the `kubeconfig` file.
3. After receiving a request with an ID token, `kube-apiserver` verifies that the token is signed by the provider configured in step 1 using keys from the JWKS endpoint. It then compares the token’s `iss` and `aud` claim values with the configuration.

How to rotate the secret of the kubernetes OAuth2 client?

The secret of the privileged `kubernetes` OAuth2 client is stored in the `kubernetes-dex-client-app-secret` Secret of the `d8-user-authn` namespace. The same value is used by the `kubeconfig-generator`, `kubeconfig-publish-api` and `kubeconfig-<slug>` OAuth2 clients, and is passed to basic-auth-proxy as `--ldap-client-secret`.

Deleting the Secret does not rotate the value: while it is still present in the module’s values, the owning hook renders the very same one back.

To rotate the secret, do the following:

1. If a GitOps tool controls the `d8-user-authn` namespace, exclude the `kubernetes-dex-client-app-secret` Secret from syncing. Otherwise the GitOps tool will restore the previous value.
2. Empty the `secret` field:

```
d8 k -n d8-user-authn patch secret kubernetes-dex-client-app-secret --type merge -p '{"data": {"secret":""}}'
```

3. Restart DKP so that the hook picks the empty field up and generates a new value:

```
d8 k -n d8-system rollout restart deployment/deckhouse
```

4. Verify that the secret value changed:

```
d8 k -n d8-user-authn get secret kubernetes-dex-client-app-secret -o jsonpath='{.data.secret}'
```

If it did not, repeat the steps 2 and 3. The module may have restored the previous before DKP was restarted.

One the secret has been rotated, the configuration of components that use it will be updated automatically and the corresponding pods will be restarted.

Kubeconfig files downloaded from the kubeconfig generator earlier carry the old client secret and stop refreshing tokens. Download these files again. ID tokens issued before the rotation keep working until they expire, which is configured via `settings.idTokenTTL` (10 minutes by default).

How to enable Kerberos (SPNEGO) SSO for LDAP?

If clients run in a corporate SSO environment (browser trusts the Dex host), Dex can accept Kerberos tickets via `Authorization: Negotiate` and log in without the password form.

Enabling Kerberos (SPNEGO) SSO for LDAP:

1. In AD/KDC, create/provision an SPN `HTTP/<dex-fqdn>` for a service account and generate a `keytab`.
2. In the cluster, create a Secret in the `d8-user-authn` namespace with the `krb5.keytab` data key.
3. In the LDAP DexProvider resource, enable `spec.ldap.kerberos`:
   - `enabled: true`
   - `keytabSecretName: <secret name>`
   - optional: `expectedRealm`, `usernameFromPrincipal`, `fallbackToPassword`

Dex will mount the `keytab` automatically and start accepting SPNEGO. A server-side `krb5.conf` is not required — tickets are validated using the keytab.

How to configure Basic Authentication for accessing Kubernetes API via LDAP?

1. Enable the `publishAPI` parameter in the `user-authn` module configuration.
2. Create a DexProvider resource of type `LDAP` and set `enableBasicAuth: true`.
3. Configure RBAC for groups obtained from LDAP.
4. Provide users with a `kubeconfig` configured for Basic Authentication (LDAP username and password).

⚠ Only one authentication provider in the cluster can have `enableBasicAuth` enabled.

A detailed example is described in the Usage section.

How is Dex protected against credential brute-forcing?

Each user is allowed no more than 20 login attempts. After the limit is exhausted, one additional attempt is added every 6 seconds.

The user-authz module

i Available with limitations in: **CE**

Available without limitations in: **BE, SE, SE+, EE**

i The module lifecycle stage: General Availability

The module generates role-based access model objects based on the standard Kubernetes RBAC mechanism. The module creates a set of cluster roles (`ClusterRole`) suitable for most user and group access management tasks.

⚠ Starting from Deckhouse Kubernetes Platform v1.64, the module features a experimental role-based access model. The current role-based access model will continue to operate but support for it will be discontinued in the future.

The experimental role-based access model is incompatible with the current one.

The module implements a role-based access model based on the standard RBAC Kubernetes

mechanism. It creates a set of cluster roles (`ClusterRole`) suitable for most user and group access management tasks.

Experimental role-based model

Unlike the current DKP role-based model, the new role-based one does not use `ClusterAuthorizationRule` and `AuthorizationRule` resources. All access rights are configured in the standard Kubernetes RBAC way, i.e., by creating `RoleBinding` or `ClusterRoleBinding` resources and specifying one of the roles prepared by the `user-authz` module in them.

The module creates special aggregated cluster roles (`ClusterRole`). By using these roles in `RoleBinding` or `ClusterRoleBinding`, you can do the following:

- Manage access to modules of a specific subsystem.

  For example, you can use the `d8:manage:networking:manager` role in `ClusterRoleBinding` to allow a network administrator to configure *network* modules (such as `cni-cilium`, `ingress-nginx`, `istio`, etc.).

- Manage access to *user* resources of modules within the namespace.

  For example, the `d8:use:role:manager` role in `RoleBinding` enables deleting/creating/editing the PodLoggingConfig resource in the namespace. At the same time, it does not grant access to the cluster-wide ClusterLoggingConfig and ClusterLogDestination resources of the `log-shipper` module, nor does it allow configuration of the `log-shipper` module itself.

The roles created by the module are divided into two classes:

- Use roles — for assigning rights to users (such as application developers) **in a specific namespace**.
- Manage roles — for assigning rights to administrators.

i

Specifics of combined access: using RoleBinding or ClusterRoleBinding together with ClusterAuthorizationRule (CAR) and AuthorizationRule (AR) for the same user.

If multitenancy mode is enabled in the cluster (the parameter `enableMultiTenancy: true`), the effective permissions of a user are the **union** of all grants that apply to that user:

- the CAR access level — only inside the namespaces allowed by its `limitNamespaces` or `namespaceSelector` parameters;
- the AuthorizationRule access level — inside the namespace the AuthorizationRule is created in;
- the permissions granted by ordinary RoleBinding resources — inside their namespaces;
- the permissions granted by ordinary ClusterRoleBinding resources (not generated by the `user-authz` module) — cluster-wide.

The namespace restrictions of a CAR limit only the permissions granted by that CAR. They do not cancel permissions granted via RoleBinding, ClusterRoleBinding, or AuthorizationRule in other namespaces; likewise, the CAR access level does not extend to those namespaces. For example, if a user has a CAR with `accessLevel: Editor` limited to the `ns-a` namespace and a RoleBinding with the `view` role in the `ns-b` namespace, the user gets Editor permissions in `ns-a` and read-only permissions in `ns-b`: the RoleBinding is not blocked by the CAR restrictions, and the CAR Editor level does not leak into `ns-b`.

In older DKP versions, the `user-authz` module's webhook rejected all requests to namespaces not listed in the user's CAR, even if the user had a RoleBinding for those namespaces. This limitation has been removed; RoleBinding and CAR can now be combined for the same user.

Use roles

⚠ The use role can only be used in the `RoleBinding` resource.

Use roles are intended to assign rights to a user **in a specific namespace**. Users refer to, for example, developers who use a cluster configured by an administrator to deploy their applications. Such users don't need to manage DKP modules or a cluster, but they need to be able to, for example, create their Ingress resources, configure application authentication, and collect logs from applications.

The use role defines permissions for accessing namespaced resources of modules and standard namespaced resources of Kubernetes (`Pod`, `Deployment`, `Secret`, `ConfigMap`, etc.).

The module creates the following use roles:

- `d8:use:role:viewer` — allows viewing standard Kubernetes resources in a specific namespace, except for Secrets and RBAC resources, as well as authenticating in the cluster;
- `d8:use:role:user` — in addition to the role `d8:use:role:viewer` it allows viewing secrets and RBAC resources in a specific namespace, connecting to pods, deleting pods (but not creating or modifying them), executing `kubectl port-forward` and `kubectl proxy`, as well as changing the number of replicas of controllers;
- `d8:use:role:manager` — in addition to the role `d8:use:role:user` it allows managing module resources (for example, `Certificate`, `PodLoggingConfig`, etc.) and standard namespaced Kubernetes resources (`Pod`, `ConfigMap`, `CronJob`, etc.) in a specific namespace;
- `d8:use:role:admin` — in addition to the role `d8:use:role:manager` it allows managing the resources `ResourceQuota`, `ServiceAccount`, `Role`, `RoleBinding`, `NetworkPolicy` in a specific namespace.

Manage roles

> ⚠ The manage role does not grant access to the namespace of user applications.
>
> The manage role grants access only to system namespaces (starting with `d8-` or `kube-`), and only to those system namespaces where the modules of the corresponding role subsystem are running.

Manage roles are intended for assigning rights to manage the entire platform or a part of it (the subsystem), but not the users applications themselves. Using a manage role you can enable, for example, a security administrator to manage cluster security modules. Thus, the security administrator will be able to configure authentication, authorization, security policies, and other respective parameters.

> ⚠ A manage role limits which modules and namespaces a subject can address, but it does not limit the privileges a subject can obtain through the modules it is allowed to manage. This matters most for the `security` subsystem: the right to manage authentication and authorization is equivalent to full control over the cluster.
>
> A subject that can manage the `user-authn` module can register an identity provider or reset the credentials of any local user. A subject that can manage the `user-authz` module can modify authorization rules. In both cases, they can obtain an identity with any privileges, including cluster administrator, so treat a `security` subsystem manage role as a cluster administrator role when planning access.

The manage role defines access rights:

- to cluster-wide Kubernetes resources;
- to manage DKP modules (`moduleConfig` resource) within the subsystem of the role, or to all DKP modules for the role `d8:manage:all:*`;
- to manage cluster-wide resources of DKP modules within the subsystem of the role, or to all resources of DKP modules for the role `d8:manage:all:*`;
- to system namespaces (starting with `d8-` or `kube-`) in which the modules of the subsystem of the role operate, or to all system namespaces for the role `d8:manage:all:*`.

The manage role name format is `d8:manage:<SUBSYSTEM>:<ACCESS_LEVEL>`, where:

- `SUBSYSTEM` is the role's subsystem. It can be one of the subsystem, or `all`, for access across all subsystems;
- `ACCESS_LEVEL` is the access level.

  Examples of manage roles:

  - `d8:manage:all:viewer` — access to view the configuration of all DKP modules (`moduleConfig` resource), their cluster-wide resources, their namespaced resources, and standard Kubernetes objects (except Secrets and RBAC resources) in all system namespaces (starting with `d8-` or `kube-`);
  - `d8:manage:all:manager` — similar to the role `d8:manage:all:viewer`, but with admin-level access, i.e., view/create/modify/delete the configuration of all DKP modules (`moduleConfig` resource), their cluster-wide resources, their namespaced resources, and standard Kubernetes objects in all system namespaces (starting with `d8-` or `kube-`);
  - `d8:manage:observability:viewer` — access to view the configuration of DKP modules (`moduleConfig` resource) from the `observability` area, their cluster-wide resources, their namespaced resources, and standard Kubernetes objects (except secrets and RBAC

resources) in the system namespaces `d8-log-shipper`, `d8-monitoring`, `d8-okmeter`, `d8-operator-prometheus`, `d8-upmeter`, `kube-prometheus-pushgateway`.

The module provides two access level for administrators:

- `viewer` — allows viewing standard Kubernetes resources, the configuration of modules (resources `moduleConfig`), cluster-wide resources of modules, and namespaced resources of modules in the module namespace;
- `manager` — in addition to the role `viewer` it allows managing standard Kubernetes resources, the configuration of modules (resources `moduleConfig`), cluster-wide resources of modules, and namespaced resources of modules in the module namespace;

Subsystems of the role-based model

Each DKP module belongs to a specific subsystem. For each subsystem, there is a set of roles with different levels of access. Roles are updated automatically when the module is enabled or disabled.

For example, for the `networking` subsystem, there are the following manage roles that can be used in `ClusterRoleBinding`:

- `d8:manage:networking:viewer`
- `d8:manage:networking:manager`

The scope of a role depends on which subsystem it belongs to:

- The scope of roles from the `all` subsystem is all system namespaces (starting with `d8-` or `kube-`) in the cluster.
- The scope of roles from other subsystems includes the namespaces in which the subsystem's modules operate (see the subsystem composition table), as well as all cluster-wide objects of the subsystem's modules.

Role-based model subsystems composition table.

Subsystem	Subsystem modules	Namespaces in which modules of a scope operate
all	All modules	All namespaces
deckhouse	• deckhouse • multitenancy-manager	• d8-multitenancy-manager • d8-system
infrastructure	• chrony • cloud-provider-aws • cloud-provider-azure • cloud-provider-dynamix • cloud-provider-gcp • cloud-provider-huaweicloud • cloud-provider-openstack • cloud-provider-vcd • cloud-provider-vsphere • cloud-provider-yandex • cloud-provider-zvirt • descheduler • istio • keepalived • local-path-provisioner • loki • node-manager • terraform-manager • vertical-pod-autoscaler	• d8-chrony • d8-cloud-instance-manager • d8-cloud-provider-aws • d8-cloud-provider-azure • d8-cloud-provider-dynamix • d8-cloud-provider-gcp • d8-cloud-provider-huaweicloud • d8-cloud-provider-openstack • d8-cloud-provider-vcd • d8-cloud-provider-vsphere • d8-cloud-provider-yandex • d8-cloud-provider-zvirt • d8-descheduler • d8-istio • d8-keepalived • d8-local-path-provisioner • d8-monitoring • d8-system • kube-system
kubernetes	• control-plane-manager • flow-schema • kube-dns • kube-proxy • priority-class	• d8-system • kube-system

Subsystem	Subsystem modules	Namespaces in which modules of a scope operate
networking	• cilium-hubble • cni-cilium • cni-flannel • cni-simple-bridge • flow-schema • ingress-nginx • istio • kube-dns • kube-proxy • metallb • network-gateway • network-policy-engine • node-local-dns • openvpn • static-routing-manager	• d8-cni-cilium • d8-cni-flannel • d8-cni-simple-bridge • d8-ingress-nginx • d8-istio • d8-metallb • d8-network-gateway • d8-openvpn • d8-static-routing-manager • d8-system • kube-system
observability	• dashboard • extended-monitoring • log-shipper • monitoring-applications • monitoring-custom • monitoring-deckhouse • monitoring-kubernetes • monitoring-kubernetes-control-plane • monitoring-ping • okmeter • operator-prometheus • prometheus • prometheus-metrics-adapter • prometheus-pushgateway • upmeter	• d8-log-shipper • d8-monitoring • d8-okmeter • d8-operator-prometheus • d8-upmeter • kube-prometheus-pushgateway
security	• admission-policy-engine • cert-manager • operator-trivy • runtime-audit-engine • secret-copier • user-authn • user-authz	• d8-admission-policy-engine • d8-cert-manager • d8-operator-trivy • d8-runtime-audit-engine • d8-user-authn

Current role-based model

Features:

- Manages user and group access control using Kubernetes RBAC;
- Manages access to scaling tools (the `allowScale` parameter of the `ClusterAuthorizationRule` or AuthorizationRule Custom Resource);
- Manages access to port forwarding (the `portForwarding` parameter of the `ClusterAuthorizationRule` or AuthorizationRule Custom Resource);
- Manages the list of allowed namespaces with a labelSelector (the `namespaceSelector` parameter of the `ClusterAuthorizationRule` Custom Resource);

In addition to the RBAC, you can use a set of high-level roles in the module:

- `User` — has access to information about all objects (including viewing pod logs) but cannot exec into containers, read secrets, and perform port-forwarding;
- `PrivilegedUser` — the same as `User` + can exec into containers, read secrets, and delete pods (and thus, restart them);
- `Editor` — is the same as `PrivilegedUser` + can create and edit all objects that are usually required for application tasks.
- `Admin` — the same as `Editor` + can delete service objects (auxiliary resources such as `ReplicaSet`, `certmanager.k8s.io/challenges` and `certmanager.k8s.io/orders`);
- `ClusterEditor` — the same as `Editor` + can manage a limited set of `cluster-wide` objects

that can be used in application tasks (`ClusterXXXMetric`, `KeepalivedInstance`, `DaemonSet`, etc.). This role is best suited for cluster operators.
- `ClusterAdmin` — the same as both `ClusterEditor` and `Admin` + can manage `cluster-wide` service objects (e.g., `MachineSets`, `Machines`, `OpenstackInstanceClasses`..., as well as `ClusterAuthorizationRule`, `ClusterRoleBindings` and `ClusterRole`). This role is best suited for cluster administrators. **Note** that since `ClusterAdmin` can edit `ClusterRoleBindings`, he can **broaden his privileges within the cluster**;
- `SuperAdmin` — can perform any actions with any objects (note that `namespaceSelector` and `limitNamespaces` restrictions remain valid).

⚠ Currently, the multi-tenancy mode (namespace-based authorization) is implemented according to a temporary scheme and **isn’t guaranteed to be entirely safe and secure**!

If a `ClusterAuthorizationRule` Custom Resource contains the `namespaceSelector` field, neither `limitNamespaces` nor `allowAccessToSystemNamespaces` are taken into consideration.

The `allowAccessToSystemNamespaces`, `namespaceSelector` and `limitNamespaces` options in the custom resource will no longer be applied if the authorization system’s webhook is unavailable for some reason. As a result, users will have access to all namespaces. After the webhook availability is restored, the options will become relevant again.

Default access list for each role

`verbs` aliases:

- read - `get`, `list`, `watch`
- read-write - `get`, `list`, `watch`, `create`, `delete`, `deletecollection`, `patch`, `update`
- write - `create`, `delete`, `deletecollection`, `patch`, `update`

Role `User`:

```
read:
    - apiextensions.k8s.io/customresourcedefinitions
    - apps/daemonsets
    - apps/deployments
    - apps/replicasets
    - apps/statefulsets
    - autoscaling.k8s.io/verticalpodautoscalers
    - autoscaling/horizontalpodautoscalers
    - batch/cronjobs
    - batch/jobs
    - configmaps
    - discovery.k8s.io/endpointslices
    - endpoints
    - events
    - events.k8s.io/events
    - extensions/daemonsets
    - extensions/deployments
    - extensions/ingresses
    - extensions/replicasets
    - extensions/replicationcontrollers
    - gateway.networking.k8s.io/backendtlspolicies
    - gateway.networking.k8s.io/gatewayclasses
    - gateway.networking.k8s.io/gateways
    - gateway.networking.k8s.io/grpcroutes
    - gateway.networking.k8s.io/httproutes
    - gateway.networking.k8s.io/listenersets
    - gateway.networking.k8s.io/referencegrants
    - gateway.networking.k8s.io/tcproutes
    - gateway.networking.k8s.io/tlsroutes
    - gateway.networking.k8s.io/udproutes
    - limitranges
    - metrics.k8s.io/nodes
    - metrics.k8s.io/pods
    - namespaces
    - networking.k8s.io/ingresses
    - networking.k8s.io/networkpolicies
    - nodes
    - persistentvolumeclaims
    - persistentvolumes
    - pods
    - pods/log
    - policy/poddisruptionbudgets
    - rbac.authorization.k8s.io/rolebindings
    - rbac.authorization.k8s.io/roles
    - replicationcontrollers
    - resourcequotas
    - serviceaccounts
    - services
    - storage.k8s.io/storageclasses
```

Role `PrivilegedUser` (includes all rules from the role `User`):

```
create:
    - pods/eviction
create,get:
    - pods/attach
    - pods/exec
delete,deletecollection:
    - pods
read:
    - secrets
```

Role `Editor` (includes all rules from the role `User`, `PrivilegedUser`):

```
read-write:
    - apps/deployments
    - apps/statefulsets
    - autoscaling.k8s.io/verticalpodautoscalers
    - autoscaling/horizontalpodautoscalers
    - batch/cronjobs
    - batch/jobs
    - configmaps
    - discovery.k8s.io/endpointslices
    - endpoints
    - extensions/deployments
    - extensions/ingresses
    - gateway.networking.k8s.io/backendtlspolicies
    - gateway.networking.k8s.io/gateways
    - gateway.networking.k8s.io/grpcroutes
    - gateway.networking.k8s.io/httproutes
    - gateway.networking.k8s.io/listenersets
    - gateway.networking.k8s.io/referencegrants
    - gateway.networking.k8s.io/tcproutes
    - gateway.networking.k8s.io/tlsroutes
    - gateway.networking.k8s.io/udproutes
    - networking.k8s.io/ingresses
    - persistentvolumeclaims
    - policy/poddisruptionbudgets
    - serviceaccounts
    - services
write:
    - secrets
```

Role `Admin` (includes all rules from the role `User`, `PrivilegedUser`, `Editor`):

```
create,patch,update:
    - pods
delete,deletecollection:
    - apps/replicasets
    - extensions/replicasets
read-write:
    - deckhouse.io/authorizationrules
    - rbac.authorization.k8s.io/rolebindings
    - rbac.authorization.k8s.io/roles
```

Role `ClusterEditor` (includes all rules from the role `User`, `PrivilegedUser`, `Editor`):

```
read:
    - rbac.authorization.k8s.io/clusterrolebindings
    - rbac.authorization.k8s.io/clusterroles
write:
    - apiextensions.k8s.io/customresourcedefinitions
    - apps/daemonsets
    - extensions/daemonsets
    - gateway.networking.k8s.io/gatewayclasses
    - storage.k8s.io/storageclasses
```

Role `ClusterAdmin` (includes all rules from the role `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`):

```
proxy:
    - nodes
read-write:
    - deckhouse.io/clusterauthorizationrules
    - namespaces
    - nodes/configz
    - nodes/healthz
    - nodes/log
    - nodes/metrics
    - nodes/pods
    - nodes/proxy
    - nodes/stats
write:
    - limitranges
    - networking.k8s.io/networkpolicies
    - rbac.authorization.k8s.io/clusterrolebindings
    - rbac.authorization.k8s.io/clusterroles
    - rbac.authorization.k8s.io/rolebindings
    - rbac.authorization.k8s.io/roles
    - resourcequotas
```

You can get additional list of access rules for module role from cluster (existing user defined rules and non-default rules from other deckhouse modules):

```
D8_ROLE_NAME=Editor
kubectl get clusterrole -A -o jsonpath="{range .items[?(@.metadata.annotations.user-authz\.deckhouse\.io/access-level=='$D8_ROLE_NAME')]}{.rules}{'\n'}{end}" | jq -s add
```

Configuration

> **i** Available with limitations in: **CE**
>
> Available without limitations in: **BE, SE, SE+, EE**

> **i** The module lifecycle stage: General Availability

> **i** **Caution!** We strongly do not recommend creating Pods and ReplicaSets – these objects are secondary and should be created by other controllers. Access to creating and modifying Pods and ReplicaSets is disabled.
>
> **Caution!** Currently, the multi-tenancy mode (namespace-based authorization) is implemented according to a temporary scheme and **isn't guaranteed to be entirely safe and secure**! The `allowAccessToSystemNamespaces`, `namespaceSelector` and `limitNamespaces` options in the custom resource will no longer be applied if the authorization system's webhook is unavailable for some reason. As a result, users will have access to all namespaces. After the webhook availability is restored, the options will become relevant again.

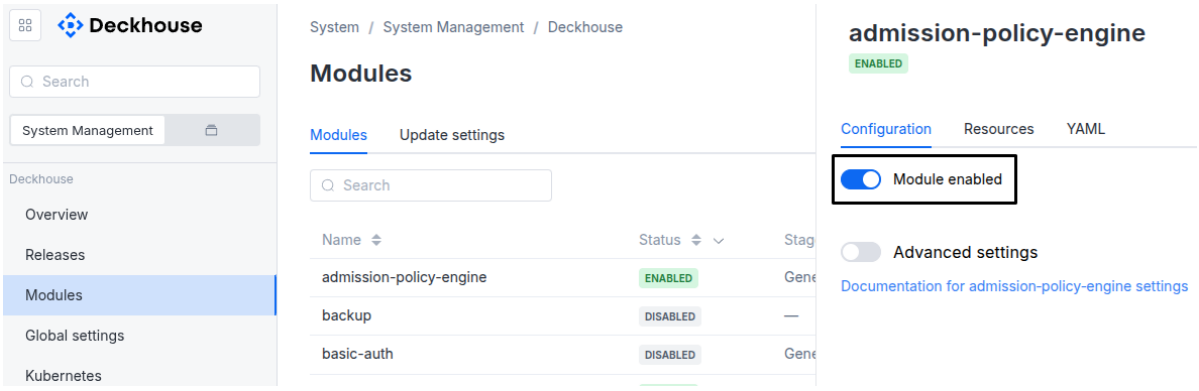
All access rights are configured using Custom Resources.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `user-authz` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:

- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

  ```
  d8 system module enable user-authz
  ```

- **Using ModuleConfig `user-authz`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `user-authz` (create it if necessary);

Example of a manifest to enable module `user-authz`:

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

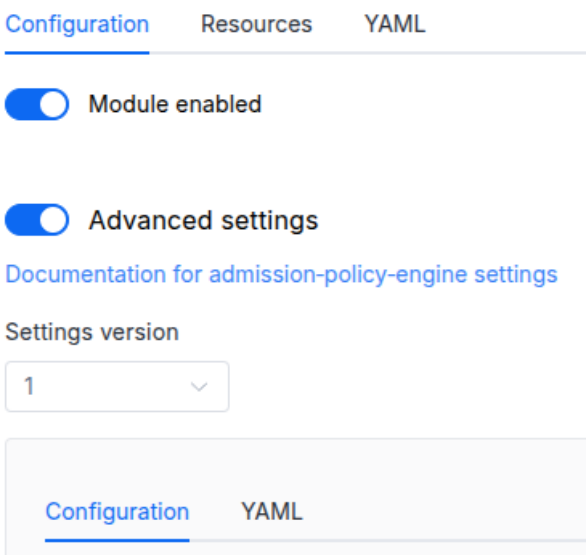
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the “System” → “System Management” → “Deckhouse” → “Modules” section, open the `user-authz` module and enable the “Advanced Settings” switch. Fill in the required fields in the “Configuration” tab or specify the module settings in YAML format on the “YAML” tab, **excluding the settings section**. Save the changes.

  Example:

  You can also edit the ModuleConfig object `user-authz` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `user-authz`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `user-authz` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc user-authz
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `user-authz` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `user-authz`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: user-authz
spec:
  version: 1
  enabled: true
  settings: # Module parameters from the "Parameters" section below.
```

Parameters

Schema version: 1

- **settings**
  object
  - **settings.controlPlaneConfigurator**
    object

    Parameters of the control-plane-manager module.

    - **settings.controlPlaneConfigurator.enabled**
      boolean

      Passes parameters for configuring `authz-webhook` to the `control-plane-manager` module (see the parameters of the control-plane-manager module).

      If this parameter is disabled, the `control-plane-manager` module assumes that Webhook-based authorization is disabled by default. In this case (if no additional settings are provided), the `control-plane-manager` module will try to delete all references to the Webhook plugin from the manifest (even if you configure the manifest manually).

      Default: `true`

      Examples:

      ```
      enabled: true
      ```

      ```
      enabled: false
      ```

  - **settings.enableMultiTenancy**
    boolean

    Available in editions: BE, SE, SE+, EE

    Enable namespace-based authorization (multi-tenancy mode).

    All the `kube-apiserver` settings required for namespace-based authorization are performed automatically by the control-plane-manager module (more details).

    Default: `false`

    Examples:

    ```
    enableMultiTenancy: true
    ```

    ```
    enableMultiTenancy: false
    ```

Custom Resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability

AuthorizationRule

Scope: Namespaced
Version: v1alpha1

Manages RBAC and authorization settings within a particular namespace.

> **i** The `AuthorizationRule` resource is used in the current role-based model and isn't used in the experimental role model.

- **spec**
  object

  Required value

  - **spec.accessLevel**
    string

    Access level:

    - `User` — has access to information about all objects (including viewing pod logs) but cannot exec into containers, read secrets, and perform port-forwarding;
    - `PrivilegedUser` — the same as `User` + can exec into containers, read secrets, and delete pods (and thus, restart them);
    - `Editor` — is the same as `PrivilegedUser` + can create and edit all objects that are usually required for application tasks;
    - `Admin` — the same as `Editor` + can delete service objects (auxiliary resources such as `ReplicaSet`, `certmanager.k8s.io/challenges` and `certmanager.k8s.io/orders`);

    Allowed values: `User`, `PrivilegedUser`, `Editor`, `Admin`

    Example:

    ```
    accessLevel: PrivilegedUser
    ```

  - **spec.allowScale**
    boolean

    Defines if scaling of Deployments and StatefulSets is allowed/not allowed.

    Default: `false`

  - **spec.portForwarding**
    boolean

    Allow/disallow the user to do `port-forwarding`.

    Default: `false`

  - **spec.subjects**
    array of objects

    Required value

    Users and/or groups to grant privileges.

    Kubernetes API reference…

    Pay attention to the following nuances if this module is used together with the user-authn module:

    - Use the user's `email` as the username to grant privileges to the specific user;
    - When specifying a group, make sure that the necessary groups are allowed to be received from the provider, i.e., they are defined in the corresponding custom resource DexProvider.
    - **spec.subjects.kind**
      string

      Required value

      Type of user identification resource.

      Allowed values: `User`, `Group`, `ServiceAccount`

Example:

```
kind: Group
```

- **spec.subjects.name**
  string

  Required value

  Resource name.

  Example:

  ```
  name: some-group-name
  ```

- **spec.subjects.namespace**
  string

  ServiceAccount namespace.

  Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

  Length: `1..63`

ClusterAuthorizationRule

Scope: Cluster

- v1
- v1alpha1

This object manages RBAC and namespace-based authorization.

The settings determine which access level is assigned to the user and/or group.

- **spec**
  object

  Required value

  - **spec.accessLevel**
    string

    Access level:

    - `User` — has access to information about all objects (including viewing pod logs) but cannot exec into containers, read secrets, and perform port-forwarding;
    - `PrivilegedUser` — the same as `User` + can exec into containers, read secrets, and delete pods (and thus, restart them);
    - `Editor` — is the same as `PrivilegedUser` + can create and edit all objects that are usually required for application tasks;
    - `Admin` — the same as `Editor` + can delete service objects (auxiliary resources such as `ReplicaSet`, `certmanager.k8s.io/challenges` and `certmanager.k8s.io/orders`);
    - `ClusterEditor` — the same as `Editor` + can manage a limited set of `cluster-wide` objects that can be used in application tasks (`ClusterXXXMetric`, `KeepalivedInstance`, `DaemonSet`, etc.). This role is best suited for cluster operators.
    - `ClusterAdmin` — the same as both `ClusterEditor` and `Admin` + can manage `cluster-wide` service objects (e.g., `MachineSets`, `Machines`, `OpenstackInstanceClasses`..., as well as `ClusterAuthorizationRule`, `ClusterRoleBindings` and `ClusterRole`). This role is best suited for cluster administrators.

      Caution! since `ClusterAdmin` can edit `ClusterRoleBindings`, he can broader his privileges within the cluster;
    - `SuperAdmin` — can perform any actions with any objects (note that `limitNamespaces` and `namespaceSelector` (see below) restrictions remain valid).

    Allowed values: `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, `ClusterAdmin`, `SuperAdmin`

    Example:

```
accessLevel: PrivilegedUser
```

- **spec.additionalRoles**
  array of objects

  Additional roles to bind for subjects.

  This parameter is reserved for emergencies. Please, use the `accessLevel` parameter instead.

  Example:

  ```
  additionalRoles:
  - apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: cluster-write-all
  - apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: cluster-read-all
  ```

  - **spec.additionalRoles.apiGroup**
    string

    Required value

    apiGroup for users.

    Example:

    ```
    apiGroup: rbac.authorization.k8s.io
    ```

  - **spec.additionalRoles.kind**
    string

    Required value

    Kind of the role.

    Allowed values: `ClusterRole`

    Example:

    ```
    kind: ClusterRole
    ```

  - **spec.additionalRoles.name**
    string

    Required value

    Name of the role.

    Example:

    ```
    name: cluster-admin
    ```

- **spec.allowAccessToSystemNamespaces**
  Deprecated
  boolean

  Available in editions: BE, SE, SE+, EE

  Allow access to System namespaces (kube-*, d8-*, loghouse, default).

  Option available **only** if the enableMultiTenancy option is enabled.

  Deprecated. Use the namespaceSelector field instead.

  Default: `false`

- **spec.allowScale**
  boolean

  Defines if scaling of Deployments and StatefulSets is allowed/not allowed.

Default: `false`

- **spec.limitNamespaces**
  Deprecated
  array of strings

  Available in editions: BE, SE, SE+, EE

  List of regex-patterns that define namespaces accessible by the user.

  The decision making process:

  - If the list is defined, then only its constituents are accessible.
  - If the list is not defined, then all namespaces are accessible (except for the system ones - see `spec.allowAccessToSystemNamespaces` below).

  Option available **only** if enableMultiTenancy option is enabled.

  Deprecated. Use the namespaceSelector field instead.

  Example:

  ```
  limitNamespaces: production-.*
  ```

  - **Element of the array**
    string
- **spec.namespaceSelector**
  object

  Available in editions: BE, SE, SE+, EE

  Defines a set of namespaces accessible by the user, using the value of the `namespaceSelector.labelSelector` parameter.

  If the `namespaceSelector` parameter is specified, the values of the `limitNamespaces` and `allowAccessToSystemNamespaces` parameters are **ignored**. If the `namespaceSelector.matchAny` field is specified, then all namespaces (including system namespaces) will be accessible. Otherwise, only the namespaces with labels matching the `namespaceSelector.labelSelector` conditions will be accessible (including system namespaces).

  If the `namespaceSelector` parameter is not specified, then the set of available namespaces is determined by the value of the `limitNamespaces` and `allowAccessToSystemNamespaces` parameters. If none of the parameters `namespaceSelector`, `limitNamespaces` and `allowAccessToSystemNamespaces` are specified, then all namespaces will be available, except for the system ones (`kube-*`, `d8-*`, `loghouse`, `default`).

  Is available **only** if the enableMultiTenancy parameter is enabled.

  - **spec.namespaceSelector.labelSelector**
    object

    Defines the label selector-based filter of the namespaces.

    Cannot be used together with the `matchAny` parameter.

    If both `matchExpressions` and `matchLabels` parameters are set, their requirements are ANDed together — they must all be satisfied in order to match. If multiple `matchExpression` conditions are provided, they all must be satisfied in order to match.

    - **spec.namespaceSelector.labelSelector.matchExpressions**
      array of objects

      An array of set-based expressions.

      Example:

```
matchExpressions:
- key: tier
  operator: In
  values:
  - production
  - staging
- key: tier
  operator: NotIn
  values:
  - production
```

- **spec.namespaceSelector.labelSelector.matchExpressions.key**
  string

  A label name.

- **spec.namespaceSelector.labelSelector.matchExpressions.operator**
  string

  A comparison operator.

  Allowed values: `In`, `NotIn`, `Exists`, `DoesNotExist`

- **spec.namespaceSelector.labelSelector.matchExpressions.values**
  array of strings

  A label value.

  - **Element of the array**
    string

    Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

    Length: `1..63`

- **spec.namespaceSelector.labelSelector.matchLabels**
  object

  A set of labels a namespace must have to match.

  Example:

```
matchLabels:
  foo: bar
  baz: who
```

- **spec.namespaceSelector.matchAny**
  boolean

  Unconditionally permits access to any namespace in the cluster, including system namespaces.

  Cannot be used together with the `labelSelector` parameter.

  Since `labelSelector` provides access to a limited number of the namespaces that match it, `matchAny` is needed in cases where access to all namespaces is to be granted.

  Allowed values: `true`

- **spec.portForwarding**
  boolean

  Allow/disallow the user to do `port-forwarding`.

  Default: `false`

- **spec.subjects**
  array of objects

  Required value

  Users and/or groups to grant privileges.

Kubernetes API reference…

Pay attention to the following nuances if this module is used together with the user-authn module:

- Use the user's `email` as the username to grant privileges to the specific user;
- When specifying a group, make sure that the necessary groups are allowed to be received from the provider, i.e., they are defined in the corresponding custom resource DexProvider.
- **spec.subjects.kind**
  string

  Required value

  Type of user identification resource.

  Allowed values: `User`, `Group`, `ServiceAccount`

  Example:

  ```
  kind: Group
  ```

- **spec.subjects.name**
  string

  Required value

  Resource name.

  Example:

  ```
  name: some-group-name
  ```

- **spec.subjects.namespace**
  string

  ServiceAccount namespace.

  Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

  Length: `1..63`

Deprecated resource. Support for the resource might be removed in a later release.

This cluster-wide object manages RBAC and authorization.

> **i** The `ClusterAuthorizationRule` resource is used in the current role-based model and isn't used in the experimental role model.

- **spec**
  object

  Required value

  - **spec.accessLevel**
    string

    Access level:

    - `User` — has access to information about all objects (including viewing pod logs) but cannot exec into containers, read secrets, and perform port-forwarding;
    - `PrivilegedUser` — the same as `User` + can exec into containers, read secrets, and delete pods (and thus, restart them);
    - `Editor` — is the same as `PrivilegedUser` + can create and edit all objects that are usually required for application tasks;
    - `Admin` — the same as `Editor` + can delete service objects (auxiliary resources such as `ReplicaSet`, `certmanager.k8s.io/challenges` and `certmanager.k8s.io/orders`);
    - `ClusterEditor` — the same as `Editor` + can manage a limited set of `cluster-wide` objects that can be used in application tasks (`ClusterXXXMetric`, `KeepalivedInstance`, `DaemonSet`, etc.). This role is best suited for cluster operators.
    - `ClusterAdmin` — the same as both `ClusterEditor` and `Admin` + can manage

`cluster-wide` service objects (e.g., `MachineSets`, `Machines`, `OpenstackInstanceClasses` ..., as well as `ClusterAuthorizationRule`, `ClusterRoleBindings` and `ClusterRole`). This role is best suited for cluster administrators.

Caution! since `ClusterAdmin` can edit `ClusterRoleBindings`, he can broader his privileges within the cluster;

- `SuperAdmin` — can perform any actions with any objects (note that `limitNamespaces` (see below) restrictions remain valid).

Allowed values: `User`, `PrivilegedUser`, `Editor`, `Admin`, `ClusterEditor`, `ClusterAdmin`, `SuperAdmin`

Example:

```
accessLevel: PrivilegedUser
```

- **spec.additionalRoles**
  array of objects

  Additional roles to bind for subjects.

  This parameter is reserved for emergencies. Please, use the `accessLevel` parameter instead.

  Example:

  ```
  additionalRoles:
  - apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: cluster-write-all
  - apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: cluster-read-all
  ```

  - **spec.additionalRoles.apiGroup**
    string

    Required value

    apiGroup for users.

    Example:

    ```
    apiGroup: rbac.authorization.k8s.io
    ```

  - **spec.additionalRoles.kind**
    string

    Required value

    Kind of the role.

    Allowed values: `ClusterRole`

    Example:

    ```
    kind: ClusterRole
    ```

  - **spec.additionalRoles.name**
    string

    Required value

    Name of the role.

    Example:

    ```
    name: cluster-admin
    ```

- **spec.allowAccessToSystemNamespaces**

Deprecated
boolean

Available in editions: BE, SE, SE+, EE

Allow access to System namespaces (kube-*, d8-*, loghouse, default).

Option available **only** if the enableMultiTenancy option is enabled.

Deprecated. Use the namespaceSelector parameter (API version `v1` of the CR) instead.

Default: `false`

- **spec.allowScale**
  boolean

  Defines if scaling of Deployments and StatefulSets is allowed/not allowed.

  Default: `false`

- **spec.limitNamespaces**
  Deprecated
  array of strings

  Available in editions: BE, SE, SE+, EE

  List of regex-patterns that define namespaces accessible by the user.

  The decision making process:

  - If the list is defined, then only its constituents are accessible.
  - If the list is not defined, then all namespaces are accessible (except for the system ones - see `spec.allowAccessToSystemNamespaces` below).

  Option available only if enableMultiTenancy option is enabled.

  Deprecated. Use the namespaceSelector parameter (API version `v1` of the CR) instead.

  Example:

  ```
  limitNamespaces: production-.*
  ```

  - **Element of the array**
    string
- **spec.portForwarding**
  boolean

  Allow/disallow the user to do `port-forwarding`.

  Default: `false`

- **spec.subjects**
  array of objects

  Required value

  Users and/or groups to grant privileges.

  Kubernetes API reference…

  Pay attention to the following nuances if this module is used together with the user-authn module:

  - Use the user's `email` as the username to grant privileges to the specific user;
  - When specifying a group, make sure that the necessary groups are allowed to be received from the provider, i.e., they are defined in the corresponding custom resource DexProvider.
  - **spec.subjects.kind**
    string

    Required value

    Type of user identification resource.

    Allowed values: `User`, `Group`, `ServiceAccount`

    Example:

```
kind: Group
```

- **spec.subjects.name**
  string

  Required value

  Resource name.

  Example:

  ```
  name: some-group-name
  ```

- **spec.subjects.namespace**
  string

  ServiceAccount namespace.

  Pattern: `[a-z0-9]([-a-z0-9]*[a-z0-9])?`

  Length: `1..63`

Usage

> **i** The module lifecycle stage: General Availability

Example of assigning rights to a cluster administrator

> **i** The example uses the experimental role-based.

To grant access to a cluster administrator, use the role `d8:manage:all:manager` in `ClusterRoleBinding`.

Example of assigning rights to a cluster administrator (User `jane`):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin-jane
subjects:
- kind: User
  name: jane.doe@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:manage:all:manager
  apiGroup: rbac.authorization.k8s.io
```

The rights that the user will get

The rights that the user will get will be limited to namespaces starting with `d8-` or `kube-`.

The user will be able to:

- View, modify, delete, and create Kubernetes resources and DKP modules;
- Modify module configurations (view, modify, delete, and create `moduleConfig` resources);
- Execute the following commands on pods and services:
  - `kubectl attach`
  - `kubectl exec`
  - `kubectl port-forward`
  - `kubectl proxy`

Example of assigning rights to a network administrator

> **i** The example uses the experimental role-based.

To grant a network administrator access to manage the network subsystem of the cluster, use the role `d8:manage:networking:manager` in `ClusterRoleBinding`.

Example of assigning rights to a network administrator (User `jane`):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: network-admin-jane
subjects:
- kind: User
  name: jane.doe@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:manage:networking:manager
  apiGroup: rbac.authorization.k8s.io
```

The rights that the user will get

The rights that the user will get will be limited to the following list of DKP module namespaces from the `networking` subsystem (the actual list depends on the list of modules included in the cluster):

- `d8-cni-cilium`
- `d8-cni-flannel`
- `d8-cni-simple-bridge`
- `d8-ingress-nginx`
- `d8-istio`
- `d8-metallb`
- `d8-network-gateway`
- `d8-openvpn`
- `d8-static-routing-manager`
- `d8-system`
- `kube-system`

The user will be able to:

- View, modify, delete, and create *standard* Kubernetes resources in the module namespace from the `networking` subsystem.

  Example of resources that the user will be able to manage (the list is not exhaustive):

  - `Certificate`
  - `CertificateRequest`
  - `ConfigMap`
  - `ControllerRevision`
  - `CronJob`
  - `DaemonSet`
  - `Deployment`
  - `Event`
  - `HorizontalPodAutoscaler`
  - `Ingress`
  - `Issuer`
  - `Job`
  - `Lease`
  - `LimitRange`
  - `NetworkPolicy`
  - `PersistentVolumeClaim`
  - `Pod`
  - `PodDisruptionBudget`
  - `ReplicaSet`
  - `ReplicationController`
  - `ResourceQuota`
  - `Role`
  - `RoleBinding`
  - `Secret`
  - `Service`

    - `ServiceAccount`
    - `StatefulSet`
    - `VerticalPodAutoscaler`
    - `VolumeSnapshot`

- View, modify, delete, and create the following resources in the modules namespace from the `networking` subsystem:

  A list of resources that the user will be able to manage:

    - `EgressGateway`
    - `EgressGatewayPolicy`
    - `FlowSchema`
    - `IngressClass`
    - `IngressIstioController`
    - `IngressNginxController`
    - `IPRuleSet`
    - `IstioFederation`
    - `IstioMulticluster`
    - `RoutingTable`

- Modify the configuration of modules (view, change, delete, and create moduleConfig resources) from the `networking` subsystem.

  List of modules that the user will be able to manage:

    - `cilium-hubble`
    - `cni-cilium`
    - `cni-flannel`
    - `cni-simple-bridge`
    - `flow-schema`
    - `ingress-nginx`
    - `istio`
    - `kube-dns`
    - `kube-proxy`
    - `metallb`
    - `network-gateway`
    - `network-policy-engine`
    - `node-local-dns`
    - `openvpn`
    - `static-routing-manager`

- Execute the following commands with pods and services in the modules namespace from the `networking` subsystem:

    - `kubectl attach`
    - `kubectl exec`
    - `kubectl port-forward`
    - `kubectl proxy`

Example of assigning administrative rights to a user within a namespace

> **i** The example uses the experimental role-based.

To assign rights to a user manage application resources within a namespace, but without the ability to configure DKP modules, use the role `d8:use:role:admin` in `RoleBinding` in the corresponding namespace.

Example of assigning rights to an application developer (User `app-developer`) in namespace `myapp`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: myapp-developer
  namespace: myapp
subjects:
- kind: User
  name: app-developer@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: d8:use:role:admin
  apiGroup: rbac.authorization.k8s.io
```

The rights that the user will get

In the `myapp` namespace, the user will be able to:

- View, modify, delete, and create Kubernetes resources. For example, the following resources (the list is not exhaustive):
  - `Certificate`
  - `CertificateRequest`
  - `ConfigMap`
  - `ControllerRevision`
  - `CronJob`
  - `DaemonSet`
  - `Deployment`
  - `Event`
  - `HorizontalPodAutoscaler`
  - `Ingress`
  - `Issuer`
  - `Job`
  - `Lease`
  - `LimitRange`
  - `NetworkPolicy`
  - `PersistentVolumeClaim`
  - `Pod`
  - `PodDisruptionBudget`
  - `ReplicaSet`
  - `ReplicationController`
  - `ResourceQuota`
  - `Role`
  - `RoleBinding`
  - `Secret`
  - `Service`
  - `ServiceAccount`
  - `StatefulSet`
  - `VerticalPodAutoscaler`
  - `VolumeSnapshot`
- View, edit, delete, and create the following DKP module resources:
  - `DexAuthenticator`
  - `DexClient`
  - `PodLogginConfig`
- Execute the following commands for pods and services:
  - `kubectl attach`
  - `kubectl exec`
  - `kubectl port-forward`
  - `kubectl proxy`

An example of `ClusterAuthorizationRule`

```
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: test-rule
spec:
  subjects:
  - kind: User
    name: some@example.com
  - kind: ServiceAccount
    name: gitlab-runner-deploy
    namespace: d8-service-accounts
  - kind: Group
    name: some-group-name
  accessLevel: PrivilegedUser
  portForwarding: true
  # This option is only available if the enableMultiTenancy parameter is set.
  allowAccessToSystemNamespaces: false
  # This option is only available if the enableMultiTenancy parameter is set.
  namespaceSelector:
    labelSelector:
      matchExpressions:
      - key: stage
        operator: In
        values:
        - test
        - review
      matchLabels:
        team: frontend
```

A `User` or `Group` subject is matched by name against the user identity in the token. A locally managed identity is indistinguishable from one asserted by an external authentication provider. Because of that, you can't create a User whose `spec.email` or a Group whose `spec.name` matches a subject of an existing ClusterAuthorizationRule. This prevents any unnoticed attempt to grant the identity privileges.

If the match is intentional, for example, if the ClusterAuthorizationRule has been written in advance, set the `user-authz.deckhouse.io/allow-authorization-rule-collision: "true"` annotation on the User or Group.

Write an email for the `User` subject in lowercase. It's recorded to the token in lowercase so a subject spelled `Admin@Example.com` won't match `admin@example.com`. A `Group`-type subject names are matched exactly, since group names are not lowercased anywhere.

This restriction is applied during the User and Group resource creation only and doesn't prevent adding existing users or groups to a ClusterAuthorizationRule. If a matching User or Group already exists, it receives the privileges immediately after it's added to the rule.

Example of granting access to all namespaces

> **i** The example refers to the current role-based model.

In multi-tenancy mode (`userAuthz.enableMultiTenancy`), namespace-based restrictions are configured using the ClusterAuthorizationRule resource fields.

Configuration options:

- **Allow access to all namespaces, including system namespaces** (`kube-*`, `d8-*`, `loghouse`, `default`, etc.). To allow access to all namespaces, including system namespaces, set `spec.namespaceSelector.matchAny: true`. If the `namespaceSelector` parameter is specified, the `limitNamespaces` and `allowAccessToSystemNamespaces` values are **ignored**.

  Example:

```
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: all-namespaces-including-system
spec:
  subjects:
  - kind: User
    name: user@example.com
  accessLevel: Editor
  namespaceSelector:
    matchAny: true
```

- **Allow access to all namespaces except system namespaces**. To grant a user access to all namespaces except system namespaces, do not specify `namespaceSelector`, `limitNamespaces`, or `allowAccessToSystemNamespaces` when creating a ClusterAuthorizationRule. For a list of system namespaces, see the CR field description.

> **i** The `SuperAdmin` access level **does not lift** the namespace restrictions specified in the `namespaceSelector` and `limitNamespaces` parameters. If you need to grant access to all namespaces, specify the scope explicitly, including via `namespaceSelector.matchAny`.

If several `ClusterAuthorizationRule` resources match the same subject, the allowed namespaces are **unioned**; the effective `accessLevel` is the **most powerful** among all matching rules. For details, refer to the FAQ.

> **⚠** Namespace restrictions from `ClusterAuthorizationRule` are enforced by the authorization webhook chain. If the webhook is unavailable, these restrictions **do not apply** until the webhook is reachable again. For more information, see the module description.

Experimental role-based model

`d8:use:role:*` roles must be bound with `RoleBinding` in a **specific** namespace — use one `RoleBinding` per namespace (or automate the process). `d8:manage:*` roles do not cover namespaces for user workloads; they only apply to system namespaces (`d8-*`, `kube-*`) within the subsystem.

Creating a user

There are two types of users in Kubernetes:

- Service accounts managed by Kubernetes via the API;
- Regular users and groups managed by some external tool that the cluster administrator configures. There are many authentication mechanisms and, accordingly, many ways to create users. Currently, two authentication methods are supported:
  - Via the `user-authn` module. The module supports the following external providers and authentication protocols: GitHub, GitLab, Atlassian Crowd, BitBucket Cloud, Crowd, LDAP, OIDC. More details — in the documentation of the `user-authn` module.
  - Via the certificates.

When issuing the authentication certificate, you need to specify the name (`CN=<name>`), the required number of groups (`O=<group>`), and sign it using the root CA of the cluster. It is this mechanism that authenticates you in the cluster when, for example, you use kubectl on a bastion node.

Creating a ServiceAccount for a machine and granting it access

You may need to create a ServiceAccount with access to the Kubernetes API when, for example, an application is deployed using a CI system.

1. Create a ServiceAccount, e.g., in the `d8-service-accounts` namespace:

```
d8 k create -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gitlab-runner-deploy
  namespace: d8-service-accounts
---
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-runner-deploy-token
  namespace: d8-service-accounts
  annotations:
    kubernetes.io/service-account.name: gitlab-runner-deploy
type: kubernetes.io/service-account-token
EOF
```

2. Grant it the necessary privileges (using the ClusterAuthorizationRule custom resource):

```
d8 k create -f - <<EOF
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: gitlab-runner-deploy
spec:
  subjects:
  - kind: ServiceAccount
    name: gitlab-runner-deploy
    namespace: d8-service-accounts
  accessLevel: SuperAdmin
  # This option is only available if the enableMultiTenancy parameter is set.
  allowAccessToSystemNamespaces: true
EOF
```

   If multitenancy is enabled in the Deckhouse configuration (via the `enableMultiTenancy` parameter), configure the namespaces the ServiceAccount has access to (via the `namespaceSelector` parameter).

3. Set the variable values (they will be used later) by running the following commands (**insert your own values**):

```
export CLUSTER_NAME=my-cluster
export USER_NAME=gitlab-runner-deploy.my-cluster
export CONTEXT_NAME=${CLUSTER_NAME}-${USER_NAME}
export FILE_NAME=kube.config
```

4. Generate the `cluster` section in the kubectl configuration file:

   Use one of the following options to access the cluster API server:

   - If there is direct access to the API server:

     1. Get a Kubernetes cluster CA certificate:

        ```
        d8 k get cm kube-root-ca.crt -o jsonpath='{ .data.ca\.crt }' > /tmp/ca.crt
        ```

     2. Generate the `cluster` section (the API server’s IP address is used for access):

```
d8 k config set-cluster $CLUSTER_NAME --embed-certs=true \
  --server=https://$(d8 k get ep kubernetes -o json | jq -rc '.subsets[0] | "\
(.addresses[0].ip):\(.ports[0].port)"') \
  --certificate-authority=/tmp/ca.crt \
  --kubeconfig=$FILE_NAME
```

- If there is no direct access to the API server, use one of the following options:
  - enable access to the API-server over the Ingress controller (the publishAPI parameter) and specify the addresses from which requests originate (the whitelistSourceRanges parameter);
  - specify addresses from which requests will originate in a separate Ingress controller (the acceptRequestsFrom parameter).
- If a non-public CA is used:

  1. Get the CA certificate from the Secret with the certificate that is used for the `api.%s` domain:

     ```
     d8 k -n d8-user-authn get secrets -o json \
       $(d8 k -n d8-user-authn get ing kubernetes-api -o jsonpath="
     {.spec.tls[0].secretName}") \
       | jq -rc '.data."ca.crt" // .data."tls.crt"' \
       | base64 -d > /tmp/ca.crt
     ```

  2. Generate the `cluster` section (an external domain and a CA for access are used):

     ```
     d8 k config set-cluster $CLUSTER_NAME --embed-certs=true \
       --server=https://$(d8 k -n d8-user-authn get ing kubernetes-api -ojson | jq
     '.spec.rules[].host' -r) \
       --certificate-authority=/tmp/ca.crt \
       --kubeconfig=$FILE_NAME
     ```

- If a public CA is used. Generate the `cluster` section (an external domain is used for access):

  ```
  d8 k config set-cluster $CLUSTER_NAME \
    --server=https://$(d8 k -n d8-user-authn get ing kubernetes-api -ojson | jq
  '.spec.rules[].host' -r) \
    --kubeconfig=$FILE_NAME
  ```

5. Generate the `user` section using the token from the Secret's ServiceAccount in the kubectl configuration file:

   ```
   d8 k config set-credentials $USER_NAME \
     --token=$(d8 k -n d8-service-accounts get secret gitlab-runner-deploy-token -o json |jq -r
   '.data["token"]' | base64 -d) \
     --kubeconfig=$FILE_NAME
   ```

6. Generate the context in the kubectl configuration file:

   ```
   d8 k config set-context $CONTEXT_NAME \
     --cluster=$CLUSTER_NAME --user=$USER_NAME \
     --kubeconfig=$FILE_NAME
   ```

7. Set the generated context as the default one in the kubectl configuration file:

   ```
   d8 k config use-context $CONTEXT_NAME --kubeconfig=$FILE_NAME
   ```

How to create a user using a client certificate

> **i** This method is recommended for system needs (authentication of kubelets, control plane components, etc.). If you need to create a “normal” user (e.g. with console access, `kubectl`, etc.), use kubeconfig generation.

To create a user with a client certificate, you can use either OpenSSL or the Kubernetes API (via the CertificateSigningRequest object).

> ⚠ Certificates issued by any of these methods cannot be revoked. If a certificate is compromised, you will need to remove all permissions for that user (this can be difficult if the user is added to any groups: you will also need to remove all relevant groups).

Creating a user using a certificate issued via OpenSSL

> ⚠ Consider security risks when using this method.
>
> The `ca.crt` and `ca.key` must not leave the master node: sign the CSR only on the master node.
>
> Signing CSRs outside the master node risks compromising the cluster root certificate.

The features of this method are:

- The client certificate must be signed on the master node to prevent the cluster certificate from being compromised.
- Access to the cluster CA key (`ca.key`) is required. Only the cluster administrator can sign certificates.

To create a user using a client certificate issued through OpenSSL, follow these steps:

1. Get the cluster’s root certificate (`ca.crt` and `ca.key`).
2. Generate the user key:

   ```
   openssl ecparam -name prime256v1 -genkey -out myuser.key
   ```

3. Create a CSR file and specify the username in it (`myuser`) and groups to which this user belongs (`mygroup1` and `mygroup2`):

   ```
   openssl req -new -key myuser.key -out myuser.csr -subj
   "/CN=myuser/O=mygroup1/O=mygroup2"
   ```

4. Upload the CSR created in the previous step (`myuser.csr` in this example) to the master node and sign it with the cluster root certificate. Example command to sign the CSR on the master node (make sure that the paths to `myuser.csr`, `ca.crt` and `ca.key` are correct for your case):

   ```
   openssl x509 -req -in myuser.csr -CA /etc/kubernetes/pki/ca.crt -CAkey
   /etc/kubernetes/pki/ca.key -CAcreateserial -out myuser.crt -days 10
   ```

Now the certificate can be specified in the config file:

```
cat << EOF
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: $(cat /etc/kubernetes/pki/ca.crt | base64 -w0)
    server: https://<cluster_host>:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: myuser
  name: myuser@kubernetes
current-context: myuser@kubernetes
kind: Config
preferences: {}
users:
- name: myuser
  user:
    client-certificate-data: $(cat myuser.crt | base64 -w0)
    client-key-data: $(cat myuser.key | base64 -w0)
EOF
```

Creating a user by issuing a certificate via the Kubernetes API

This is a more secure method because a special kubernetes API is used to sign the certificate.

The features of this method are:

- API-based certificate signing: CSRs are processed through the Kubernetes API without requiring access to the CA's private key (`ca.key`).
- Not only the cluster administrator can issue client certificates. The right to create CSRs and sign them can be assigned to a specific user.

To create a user using a client certificate issued through the Kubernetes API, follow these steps:

1. Generate the user key:

   ```
   openssl ecparam -name prime256v1 -genkey -out myuser.key
   ```

2. Create a CSR file and specify in it the username (`myuser`) and groups to which this user belongs (`mygroup1` and `mygroup2`):

   ```
   openssl req -new -key myuser.key -out myuser.csr -subj
   "/CN=myuser/O=mygroup1/O=mygroup2"
   ```

3. Create a manifest for the CertificateSigningRequest object and save it to a file (`csr.yaml` in this example):

   > **i** In the `request` field, specify the contents of the CSR created in the previous step, encoded in Base64.

```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
name: demo-client-cert
spec:
  request: # CSR in Base64
  signerName: "kubernetes.io/kube-apiserver-client"
  expirationSeconds: 7200
  usages:
  - "digital signature"
  - "client auth"
```

4. Apply the manifest to create a certificate signing request:

   ```
   d8 k apply -f csr.yaml
   ```

5. Check that the certificate has been approved and issued:

   ```
   d8 k get csr demo-client-cert
   ```

   If the certificate is issued, it will have the value `Approved,Issued` in the `CONDITION` column. Example output:

   ```
    NAME               AGE     SIGNERNAME                              REQUESTOR
   REQUESTEDDURATION   CONDITION
    demo-client-cert   8m24s   kubernetes.io/kube-apiserver-client   kubernetes-admin   120m
   Approved,Issued
   ```

   If the certificate is not automatically verified, verify it:

   ```
   d8 k certificate approve demo-client-cert
   ```

   Then, confirm that the certificate has been successfully approved.

6. Extract the encoded certificate from the CSR named `demo-client-cert`, decode it from Base64 and save it to the file (`myuser.crt` in this example) created in step 2:

   ```
   d8 k get csr demo-client-cert -ojsonpath="{.status.certificate}" | base64 -d >
   myuser.crt
   ```

Now the certificate can be specified in the config file:

```
cat << EOF
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: $(cat /etc/kubernetes/pki/ca.crt | base64 -w0)
    server: https://<cluster_host>:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: myuser
  name: myuser@kubernetes
current-context: myuser@kubernetes
kind: Config
preferences: {}
users:
- name: myuser
  user:
    client-certificate-data: $(cat myuser.crt | base64 -w0)
    client-key-data: $(cat myuser.key | base64 -w0)
EOF
```

Granting access to the created user

To grant access to the created user, create a `ClusterAuthorizationRule'.

Example of a `ClusterAuthorizationRule`:

```
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: myuser
spec:
  subjects:
  - kind: User
    name: myuser
  accessLevel: PrivilegedUser
  portForwarding: true
```

Configuring kube-apiserver for multi-tenancy mode

The multi-tenancy mode, which allows you to restrict access to namespaces, is enabled by the enableMultiTenancy module's parameter.

Working in multi-tenancy mode requires enabling the Webhook authorization plugin and configuring a `kube-apiserver`. All actions necessary for the multi-tenancy mode are performed **automatically** by the control-plane-manager module; no additional steps are required.

Changes to the `kube-apiserver` manifest that will occur after enabling multi-tenancy mode:

- The authorization chain will be configured using the structured authorization configuration (`--authorization-config=/etc/kubernetes/deckhouse/extra-files/authorization-config.yaml`) with `Node`, `Webhook`, and `RBAC` authorizers;
- The `kubeconfig` for the authorization webhook will be created at `/etc/kubernetes/deckhouse/extra-files/webhook-config.yaml`;
- The `volumeMounts` parameter will be added:

  ```
  - name: authorization-webhook-config
    mountPath: /etc/kubernetes/authorization-webhook-config.yaml
    readOnly: true
  ```

- The `volumes` parameter will be added:

```
- name: authorization-webhook-config
  hostPath:
    path: /etc/kubernetes/authorization-webhook-config.yaml
    type: FileOrCreate
```

How do I check that a user has access?

Execute the command below with the following parameters:

- `resourceAttributes` (the same as in RBAC) - target resources;
- `user` - the name of the user;
- `groups` - user groups;

> **i** You can use Dex logs to find out groups and a username if this module is used together with the `user-authn` module (`d8 k -n d8-user-authn logs -l app=dex`); logs available only if the user is authorized.

```
cat  <<EOF | 2>&1 d8 k  create --raw
/apis/authorization.k8s.io/v1/subjectaccessreviews -f - | jq .status
{
  "apiVersion": "authorization.k8s.io/v1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "",
      "verb": "watch",
      "version": "v1",
      "resource": "pods"
    },
    "user": "system:kube-controller-manager",
    "groups": [
      "Admins"
    ]
  }
}
EOF
```

You will see if access is allowed and what role is used:

```
{
  "allowed": true,
  "reason": "RBAC: allowed by ClusterRoleBinding \"system:kube-controller-manager\"
of ClusterRole \"system:kube-controller-manager\" to User \"system:kube-controller-
manager\""
}
```

If the **multitenancy** mode is enabled in your cluster, you need to perform another check to be sure that the user has access to the namespace:

```
cat  <<EOF | 2>&1 d8 k --kubeconfig /etc/kubernetes/deckhouse/extra-files/webhook-
config.yaml create --raw / -f - | jq .status
{
  "apiVersion": "authorization.k8s.io/v1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "",
      "verb": "watch",
      "version": "v1",
      "resource": "pods"
    },
    "user": "system:kube-controller-manager",
    "groups": [
      "Admins"
    ]
  }
}
EOF
```

```
{
  "allowed": false
}
```

The `allowed: false` message means that the webhook doesn’t block access. In case of webhook denying the request, you will see, e.g., the following message:

```
{
  "allowed": false,
  "denied": true,
  "reason": "making cluster scoped requests for namespaced resources are not
allowed"
}
```

Customizing rights of high-level roles

If you want to grant more privileges to a specific high-level role, you only need to create a ClusterRole with the `user-authz.deckhouse.io/access-level: <AccessLevel>` annotation.

An example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    user-authz.deckhouse.io/access-level: Editor
  name: user-editor
rules:
- apiGroups:
  - kuma.io
  resources:
  - trafficroutes
  - trafficroutes/finalizers
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
- apiGroups:
  - flagger.app
  resources:
  - canaries
  - canaries/status
  - metrictemplates
  - metrictemplates/status
  - alertproviders
  - alertproviders/status
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
```

FAQ

> **i** The module lifecycle stage: General Availability

How do I create a user?

Creating a user.

How do I limit user rights to specific namespaces?

To limit a user's rights to specific namespaces in the experimental role-based model, use `RoleBinding` with the use role that has the appropriate level of access. Example....

In the current role-based model, use the `namespaceSelector` or `limitNamespaces` (deprecated) parameters in the `ClusterAuthorizationRule` CR.

What if there are two ClusterAuthorizationRules matching to a single user?

In the example, the user `jane.doe@example.com` is in the `administrators` group. There are two cluster authorization rules:

```
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: jane
spec:
  subjects:
    - kind: User
      name: jane.doe@example.com
  accessLevel: User
  namespaceSelector:
    labelSelector:
      matchLabels:
        env: review
---
apiVersion: deckhouse.io/v1
kind: ClusterAuthorizationRule
metadata:
  name: admin
spec:
  subjects:
  - kind: Group
    name: administrators
  accessLevel: ClusterAdmin
  namespaceSelector:
    labelSelector:
      matchExpressions:
      - key: env
        operator: In
        values:
        - prod
        - stage
```

1. `jane.doe@example.com` has the right to get and list any objects in the namespaces labeled `env=review`
2. `Administrators` can get, edit, list, and delete objects on the cluster level and in the namespaces labeled `env=prod` and `env=stage`.

Because `Jane Doe` matches two rules, some calculations will be made:

- `Jane Doe` will have the most powerful accessLevel across all matching rules — `ClusterAdmin`.
- The `namespaceSelector` options will be combined, so that Jane will have access to all the namespaces labeled with `env` label of the following values: `review`, `stage`, or `prod`.

> ⚠ If there is a rule without the `namespaceSelector` option and `limitNamespaces` deprecated option, it means that all namespaces are allowed excluding system namespaces, which will affect the resulting limit namespaces calculation.

How do I extend a role or create a new one?

The experimental role model is based on the aggregation principle; it compiles smaller roles into larger ones, thus providing easy ways to enhance the model with custom roles.

Creating a new role subsystem

Suppose that the current subsystems do not fit the role distribution in the company. You need to create a new subsystem that includes roles from the `deckhouse` subsystem, the `kubernetes` subsystem and the user-authn module.

To meet this need, create the following role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: custom:manage:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: admin
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/level: subsystem
    rbac.deckhouse.io/subsystem: custom
    rbac.deckhouse.io/aggregate-to-all-as: manager
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-kubernetes-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        module: user-authn
rules: []
```

The labels for the new role listed at the top suggest that:

- The hook will use this use role:

  ```
  rbac.deckhouse.io/use-role: admin
  ```

- The role must be treated as a managed one:

  ```
  rbac.deckhouse.io/kind: manage
  ```

  > **i** Note that this label is mandatory.

- The role is a subsystem one, and it shall be handled accordingly:

  ```
  rbac.deckhouse.io/level: subsystem
  ```

- There is a subsystem for which the role is responsible:

  ```
  rbac.deckhouse.io/subsystem: custom
  ```

- The `manage:all` role can aggregate this role:

  ```
  rbac.deckhouse.io/aggregate-to-all-as: manager
  ```

Then there are selectors that implement aggregation:

- This one aggregates the manager role from the `deckhouse` subsystem:

  ```
  rbac.deckhouse.io/kind: manage
  rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
  ```

- This one aggregates all the rules defined for the user-authn module:

  ```
  rbac.deckhouse.io/kind: manage
  module: user-authn
  ```

This way, your role will combine permissions of the `deckhouse` subsystem, `kubernetes` subsystem, and the user-authn module.

Notes:

- There are no restrictions on role name, but we recommend following the same pattern for the sake of readability.
- Use-roles will be created in aggregate subsystems and the module namespace, the role type is specified by the label.

Extending the custom role

Suppose a new cluster CRD object, MySuperResource, has been created in the cluster (a manage role example), and you need to extend the custom role from the example above to include the permissions to interact with this resource.

First, you have to add a new selector to the role:

```
rbac.deckhouse.io/kind: manage
rbac.deckhouse.io/aggregate-to-custom-as: manager
```

This selector would enable roles to be aggregated to a new subsystem by specifying this label. After adding the new selector, the role will look as follows:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: custom:manage:mycustom:manager
  labels:
    rbac.deckhouse.io/use-role: admin
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/level: subsystem
    rbac.deckhouse.io/subsystem: custom
    rbac.deckhouse.io/aggregate-to-all-as: manager
aggregationRule:
  clusterRoleSelectors:
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-kubernetes-as: manager
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        module: user-authn
    - matchLabels:
        rbac.deckhouse.io/kind: manage
        rbac.deckhouse.io/aggregate-to-custom-as: manager
rules: []
```

Next, you need to create a new role and define permissions for the new resource, e. g., the read-only permission:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-custom-as: manager
    rbac.deckhouse.io/kind: manage
  name: custom:manage:permission:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

The role will update the subsystem role to include its rights, so that the role bearer will be able to view the new object.

Notes:

- There are no restrictions on capability names, but we recommend following the same pattern for the sake of readability.

Extending the existing manage subsystem roles

To extend an existing role, follow the procedure outlined in the section above. Be sure to change the labels and the role name!

For example, here’s how you can extend the manager role from the `deckhouse` (`d8:manage:deckhouse:manager`) subsystem:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    rbac.deckhouse.io/kind: manage
  name: custom:manage:permission:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

This way, the new role will extend the `d8:manage:deckhouse:manager` role.

Extending manage subsystem roles and adding a new namespace

If you need to create a new namespace (to create a use role in it by the hook), you only need to add one label:

```
"rbac.deckhouse.io/namespace": namespace
```

This label instructs the hook to create a use role in this namespace:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-deckhouse-as: manager
    rbac.deckhouse.io/kind: manage
    rbac.deckhouse.io/namespace: namespace
  name: custom:manage:permission:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

The hook monitors `ClusterRoleBinding`, and when creating a bindings, it loops through all the manage roles to find all the aggregated roles by checking the aggregation rule. It then fetches the namespace from the `rbac.deckhouse.io/namespace` label and creates a use role in that namespace.

Extending the existing use roles

If the resource belongs to a namespace, you need to extend the use role instead of the manage role. The only difference is the labels and the name:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.deckhouse.io/aggregate-to-kubernetes-as: user
    rbac.deckhouse.io/kind: use
  name: custom:use:capability:mycustom:superresource:view
rules:
- apiGroups:
  - mygroup.io
  resources:
  - mysuperresources
  verbs:
  - get
  - list
  - watch
```

This role will be added to the `d8:use:role:user:kubernetes` role.

The vertical-pod-autoscaler module

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

Vertical Pod Autoscaler (VPA) is an infrastructure service that allows you to avoid setting exact resource requests for a container if their values are not known in advance. VPA can automatically adjust the CPU and memory reservations (providing that the corresponding mode is enabled) based on the actual resource consumption (as shown by the Prometheus data). Also, VPA can recommend values for resource requests and limits without updating them automatically.

VPA has the following operating modes:

- `"Auto"` (default) — currently the `Auto` and `Recreate` modes do the same thing. However, when Pod in-place resource update appears in Kubernetes, this mode will do exactly that.
- `"Recreate"` — the mode allows VPA to change resources of running pods (restart them while

running). In case of one pod running (`replicas: 1`), this will lead to the service being unavailable during the restart. In this mode, VPA does not recreate pods that were created without a controller.
- `"Initial"` — VPA modifies Pod resources only when Pods are started (but not during operation).
- `"Off"` — VPA does not take any action to update the resource requests for the running containers. In this case, if VPA is running in this mode, you can see what resource values it recommends (`d8 k describe vpa <vpa-name>`).

⚠ The `Auto` mode is deprecated. It will no longer be supported in future DKP versions. Switch to one of the supported operating modes.

VPA limitations:

- Updating the resources of running Pods is currently experimental. The Pod is recreated each time VPA updates its `resource requests`. The Pod can be scheduled to another node.
- VPA **should not be used concurrently with CPU and memory-based HPA**. However, you can use VPA together with HPA for custom/external metrics.
- VPA notices almost all `out-of-memory` events, but that does not guarantee its response.
- VPA performance has not been tested for huge clusters.
- VPA recommendations may exceed the available resources in the cluster. That can lead to **Pods becoming Pending**.
- Using multiple VPAs for the same Pod can lead to undefined behavior.
- If VPA is deleted or “turned off” (the `Off` mode), the changes made by VPA earlier are not reset (the most recent value set is kept). It may lead to confusion due to the difference between resource values in Helm/controller and the actual resources of Pods (it may be perceived as if they “came from out of nowhere”).

i We highly recommend using Pod Disruption Budget with VPA.

Grafana dashboard

Displayed on dashboards:

- `Main / Namespace`, `Main / Namespace / Controller`, `Main / Namespace / Controller / Pod` — the `VPA type` column shows the value of `updatePolicy.updateMode`;
- `Main / Namespaces` — the `VPA %` column shows the percentage of VPA-enabled Pods.

Vertical Pod Autoscaler Architecture

VPA consists of 3 components:

- `Recommender` — this component monitors the current resource consumption (by making requests to the Metrics API implemented in the `prometheus-metrics-adapter` module) as well as consumption history (by making requests to Trickster caching proxy). As its name suggests, the component provides CPU and memory recommendations for containers.
- `Updater` — this component checks if the Pods have correct resources set and, if not, kills them so that they can be recreated by their controllers with the updated resource requests.
- `Admission Plugin` — this component sets the correct resource requests on new Pods (either just created or recreated by their controller due to Updater’s activity).

When the Updater component changes resource values, Pods are evicted using the Eviction API. Thus, the `Pod Disruption Budget` is taken into account for the Pods being updated.

Third-party components

List of third-party software used in the `vertical-pod-autoscaler` module:

- Kubernetes Autoscaler 1.6.0

  License: Apache License 2.0

  A autoscaling-related components for Kubernetes.

The vertical-pod-autoscaler: configuration

> The module lifecycle stage: General Availability
> The module has requirements for installation

VPA works directly with the Pod (instead of the Pod controller) by measuring and changing its containers' parameters. Configuring is performed using the `VerticalPodAutoscaler` Custom Resource.

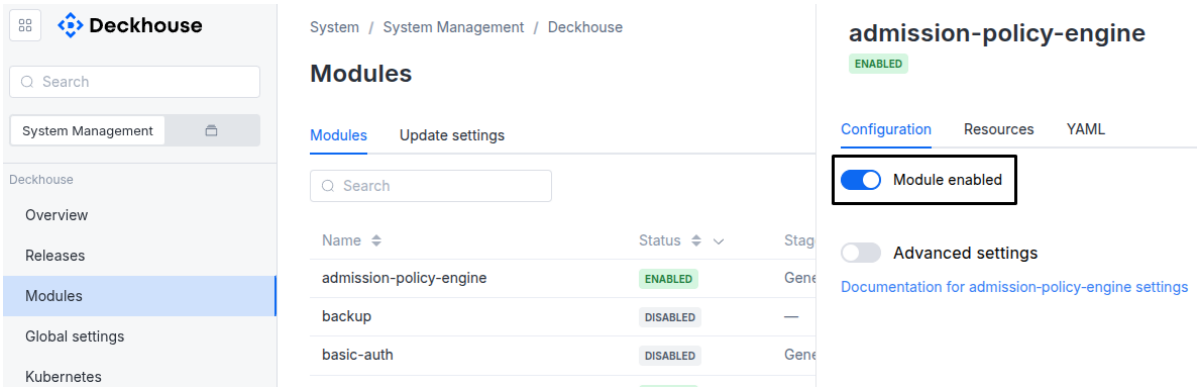
The module generally requires no configuration and only has the `nodeSelector/tolerations` settings.

The module is **enabled** by default in the following bundles: `Default`, `Managed`. The module is **disabled** by default in the `Minimal` bundle.

You may explicitly enable or disable the module in one of the following ways:

- **Via Deckhouse web UI.** In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `vertical-pod-autoscaler` module and enable (or disable) the "Module enabled" toggle. Save changes.

  Example:



- **Via Deckhouse CLI (d8).**

  Use the d8 system module enable command for enabling, or d8 system module disable command for disabling the module (you need Deckhouse CLI (d8), configured to work with the cluster).

  Example of enabling the module:

```
d8 system module enable vertical-pod-autoscaler
```

- **Using ModuleConfig `vertical-pod-autoscaler`.**

  Set `spec.enabled` to `true` or `false` in ModuleConfig `vertical-pod-autoscaler` (create it if necessary);

  Example of a manifest to enable module `vertical-pod-autoscaler`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  enabled: true
```

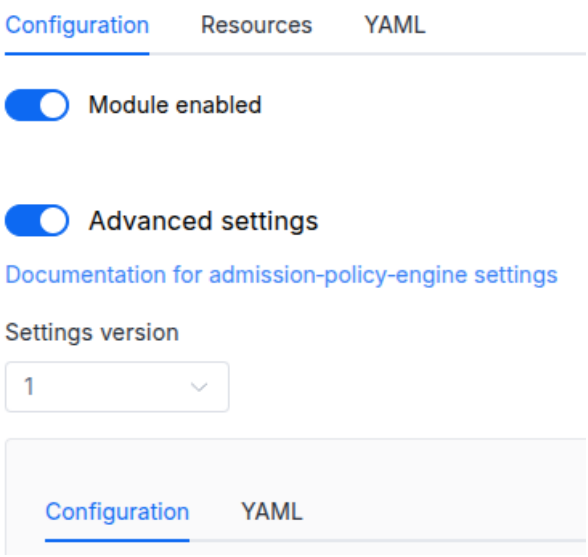
How to configure the module...

You can configure the module in one of the following ways:

- **Via Deckhouse web UI.**

  In the "System" → "System Management" → "Deckhouse" → "Modules" section, open the `vertical-pod-autoscaler` module and enable the "Advanced Settings" switch. Fill in the required fields in the "Configuration" tab or specify the module settings in YAML format on the "YAML" tab, **excluding the settings section**. Save the changes.

  Example:



You can also edit the ModuleConfig object `vertical-pod-autoscaler` on the “YAML” tab in the module settings window (“System” → “System Management” → “Deckhouse” → “Modules”, open the module `vertical-pod-autoscaler`) by specifying the schema version in the `spec.version` parameter and the necessary module parameters in the `spec.settings` section.

- **Via Deckhouse CLI (d8)** (requires Deckhouse CLI (d8) configured to work with the cluster).

  Edit the existing ModuleConfig `vertical-pod-autoscaler` (for more details on configuring Deckhouse, see the documentation) by executing the following command:

  ```
  d8 k edit mc vertical-pod-autoscaler
  ```

  Make the necessary changes in the `spec.settings` section. If necessary, specify the schema version in the `spec.version` parameter. Save the changes.

  You can also create a file with manifest for ModuleConfig `vertical-pod-autoscaler` using the example below. Fill in the `spec.settings` section with the required module parameters. If necessary, specify the schema version in the `spec.version` parameter.

  Apply the manifest using the following command (indicate the manifest file name):

  ```
  d8 k apply -f <FILENAME>
  ```

  Example of a manifest for ModuleConfig `vertical-pod-autoscaler`:

  ```
  apiVersion: deckhouse.io/v1alpha1
  kind: ModuleConfig
  metadata:
    name: vertical-pod-autoscaler
  spec:
    version: 1
    enabled: true
    settings: # Module parameters from the "Parameters" section below.
  ```

Requirements

To the versions of other modules:

- `prometheus`: any version.
- `prometheus-metrics-adapter`: any version.

Parameters

Schema version: 1

- **settings**
  object
  - **settings.nodeSelector**

object

The same as in the pods' `spec.nodeSelector` parameter in Kubernetes.

If the parameter is omitted or `false`, it will be determined automatically.

- **settings.tolerations**
  array of objects

  The same as in the pods' `spec.tolerations` parameter in Kubernetes;

  If the parameter is omitted or `false`, it will be determined automatically.

  - **settings.tolerations.effect**
    string
  - **settings.tolerations.key**
    string
  - **settings.tolerations.operator**
    string
  - **settings.tolerations.tolerationSeconds**
    integer
  - **settings.tolerations.value**
    string

Custom resources

> **i** The Deckhouse Kubernetes Platform installs CRDs but does not remove them when a module is disabled. If you no longer need the created CRDs, delete them.

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

VerticalPodAutoscaler

Short names: vpa

Scope: Namespaced

- v1
- v1beta2

Configuration for a vertical pod autoscaler, which automatically manages pod resources based on historical and real time resource utilization.

- **apiVersion**
  string

  Defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values.

  More information in the documentation.

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase.

  More information in the documentation.

- **metadata**
  object
- **spec**
  object

  Required value

  Specification of the behavior of the autoscaler.

  More information in the documentation.

  - **spec.recommenders**

array of objects

Recommender responsible for generating recommendation for this object. List should be empty (then the default recommender will generate the recommendation) or contain exactly one recommender.

VerticalPodAutoscalerRecommenderSelector points to a specific Vertical Pod Autoscaler recommender. In the future it might pass parameters to the recommender.

- **spec.recommenders.name**
  string

  Required value

  Name of the recommender responsible for generating recommendation for this object.

- **spec.resourcePolicy**
  object

  Controls how the autoscaler computes recommended resources. The resource policy may be used to set constraints on the recommendations for individual containers.

  If any individual containers need to be excluded from getting the VPA recommendations, then it must be disabled explicitly by setting mode to `Off` under `containerPolicies`.

  If not specified, the autoscaler computes recommended resources for all containers in the pod, without additional constraints.

  - **spec.resourcePolicy.containerPolicies**
    array of objects

    Per-container resource policies.

    Controls how autoscaler computes the recommended resources for a specific container.

    - **spec.resourcePolicy.containerPolicies.containerName**
      string

      Name of the container or `DefaultContainerResourcePolicy`, in which case the policy is used by the containers that don’t have their own policy specified.

    - **spec.resourcePolicy.containerPolicies.controlledResources**
      array of strings

      Specifies the type of recommendations that will be computed (and possibly applied) by VPA. If not specified, the default of (`ResourceCPU`, `ResourceMemory`) will be used.

      Default: `["cpu","memory"]`

      - **Element of the array**
        string

        ResourceName is the name identifying various resources in a ResourceList.

    - **spec.resourcePolicy.containerPolicies.controlledValues**
      string

      Specifies which resource values should be controlled.

      Default: `RequestsAndLimits`

      Allowed values: `RequestsAndLimits`, `RequestsOnly`

    - **spec.resourcePolicy.containerPolicies.maxAllowed**
      object

      Specifies the maximum amount of resources that will be recommended for the container. The default is no maximum.

    - **spec.resourcePolicy.containerPolicies.minAllowed**
      object

      Specifies the minimal amount of resources that will be recommended for the container. The default is no minimum.

    - **spec.resourcePolicy.containerPolicies.mode**
      string

Whether autoscaler is enabled for the container.

Default: `Auto`

Allowed values: `Auto`, `Off`

- **spec.scope**
  string

  Scope controls additional recommendation partitioning for DaemonSet targets.

  The value must be a node label key (for example: `node.kubernetes.io/instance-type`).

- **spec.targetRef**
  object

  Required value

  Points to the controller managing the set of pods for the autoscaler to control — e.g. Deployment, StatefulSet.

  VerticalPodAutoscaler can be targeted at controller implementing scale subresource (the pod set is retrieved from the controller's `ScaleStatus`) or some well known controllers (e.g. for DaemonSet the pod set is read from the controller's spec).

  If VerticalPodAutoscaler cannot use specified target it will report `ConfigUnsupported` condition. Note that VerticalPodAutoscaler does not require full implementation of scale subresource — it will not use it to modify the replica count. The only thing retrieved is a label selector matching pods grouped by the target resource.

  - **spec.targetRef.apiVersion**
    string

    API version of the referent.

  - **spec.targetRef.kind**
    string

    Required value

    Kind of the referent.

    More info.

  - **spec.targetRef.name**
    string

    Required value

    Name of the referent.

    More info.

- **spec.updatePolicy**
  object

  Describes the rules on how changes are applied to the pods. If not specified, all fields in the `PodUpdatePolicy` are set to their default values.

  - **spec.updatePolicy.evictionRequirements**
    array of objects

    EvictionRequirements is a list of EvictionRequirements that need to evaluate to true in order for a Pod to be evicted. If more than one EvictionRequirement is specified, all of them need to be fulfilled to allow eviction.

    EvictionRequirement defines a single condition which needs to be true in order to evict a Pod.

    - **spec.updatePolicy.evictionRequirements.changeRequirement**
      string

      Required value

      EvictionChangeRequirement refers to the relationship between the new target recommendation for a Pod and its current requests, what kind of change is necessary for the Pod to be evicted.

      Allowed values: `TargetHigherThanRequests`, `TargetLowerThanRequests`

        - **spec.updatePolicy.evictionRequirements.resources**
          array of strings

          Required value

          Resources is a list of one or more resources that the condition applies to. If more than one resource is given, the EvictionRequirement is fulfilled if at least one resource meets `changeRequirement`.

          - **Element of the array**
            string

            ResourceName is the name identifying various resources in a ResourceList.

      - **spec.updatePolicy.minReplicas**
        integer

        Minimal number of replicas which need to be alive for Updater to attempt pod eviction (pending other checks like PDB). Only positive values are allowed. Overrides global '–min-replicas' flag.

      - **spec.updatePolicy.updateMode**
        string

        The operation mode of the VPA controller.

        - `Auto` — currently, Auto and Recreate modes do the same thing. This mode is reserved for the Pod inplace resource update in Kubernetes.
        - `Recreate` — this mode allows VPA to modify resource requirements of the running pods (i.e., restart them during operation). This mode might result in temporary unavailability of the service (due to restart) if you have just one replica (replicas: 1). In this mode, VPA does not recreate pods that were not created by a controller.
        - `Initial` — VPA modifies pod resources only when pods are started (but not when pods are running).
        - `Off` — VPA does not take any action to update the resource requests for the running containers. Autoscaler calculates recomendations and stores them in the status field of the VPA object. You can browse VPA recommendations using the following command: `d8 k describe vpa <vpa-name>`.
        - `InPlaceOrRecreate` (GA) — supported in Kubernetes 1.33 and later. The VPA attempts to update pod resources without a restart; if that is not possible, it recreates the pod.

        Default: `InPlaceOrRecreate`

        Allowed values: `Off`, `Initial`, `Recreate`, `Auto`, `InPlaceOrRecreate`

- **status**
  object

  Current information about the autoscaler.

  - **status.conditions**
    array of objects

    Set of conditions required for this autoscaler to scale its target, and indicates whether or not those conditions are met.

    Describes the state of a VerticalPodAutoscaler at a certain point.

    - **status.conditions.lastTransitionTime**
      string

      Last time the condition transitioned from one status to another.

    - **status.conditions.message**
      string

      A human-readable explanation containing details about the transition.

    - **status.conditions.reason**
      string

      Reason for the condition's last transition.

    - **status.conditions.status**
      string

Required value

Status of the condition (`True`, `False`, Unknown).

- **status.conditions.type**
  string

  Required value

  Describes the current condition.

- **status.groups**
  array of objects

  Contains grouped recommendations by scope value when `spec.scope` is used.

  - **status.groups.containerRecommendations**
    array of objects

    Resources recommended by autoscaler for each container in this group.

    - **status.groups.containerRecommendations.containerName**
      string

      Name of the container.

    - **status.groups.containerRecommendations.lowerBound**
      object

      Minimum recommended amount of resources. Observes `ContainerResourcePolicy`. This amount is not guaranteed to be sufficient for the application to operate in a stable way, however running with less resources is likely to have significant impact on performance/availability.

    - **status.groups.containerRecommendations.scope**
      string

      Scope binds this recommendation to a specific scope value in form “ =".

    - **status.groups.containerRecommendations.target**
      object

      Required value

      Recommended amount of resources. Observes `ContainerResourcePolicy`.

    - **status.groups.containerRecommendations.uncappedTarget**
      object

      The most recent recommended resources target computed by the autoscaler for the controlled pods, based only on actual resource usage, not taking into account the `ContainerResourcePolicy`.

      May differ from the Recommendation if the actual resource usage causes the target to violate the `ContainerResourcePolicy` (lower than MinAllowed or higher that MaxAllowed). Used only as status indication, will not affect actual resource assignment.

    - **status.groups.containerRecommendations.upperBound**
      object

      Maximum recommended amount of resources.

      Observes `ContainerResourcePolicy`. Any resources allocated beyond this value are likely wasted. This value may be larger than the maximum amount of application is actually capable of consuming.

  - **status.groups.scopeValue**
    string

    Required value

    Scope value of `spec.scope` label key for this recommendation group.

- **status.recommendation**
  object

  The most recently computed amount of resources recommended by the autoscaler for the

controlled pods.

- **status.recommendation.containerRecommendations**
  array of objects

  Resources recommended by the autoscaler for each container.

  Recommendation of resources computed by autoscaler for a specific container. Respects the container resource policy if present in the spec. In particular the recommendation is not produced for containers with `ContainerScalingMode` set to 'Off'.

  - **status.recommendation.containerRecommendations.containerName**
    string

    Name of the container.

  - **status.recommendation.containerRecommendations.lowerBound**
    object

    Minimum recommended amount of resources. Observes `ContainerResourcePolicy`. This amount is not guaranteed to be sufficient for the application to operate in a stable way, however running with less resources is likely to have significant impact on performance/availability.

  - **status.recommendation.containerRecommendations.scope**
    string

    Scope binds this recommendation to a specific scope value in form " =".

  - **status.recommendation.containerRecommendations.target**
    object

    Required value

    Recommended amount of resources. Observes `ContainerResourcePolicy`.

  - **status.recommendation.containerRecommendations.uncappedTarget**
    object

    The most recent recommended resources target computed by the autoscaler for the controlled pods, based only on actual resource usage, not taking into account the `ContainerResourcePolicy`. May differ from the Recommendation if the actual resource usage causes the target to violate the `ContainerResourcePolicy` (lower than MinAllowed or higher that MaxAllowed). Used only as status indication, will not affect actual resource assignment.

  - **status.recommendation.containerRecommendations.upperBound**
    object

    Maximum recommended amount of resources. Observes `ContainerResourcePolicy`. Any resources allocated beyond this value are likely wasted. This value may be larger than the maximum amount of application is actually capable of consuming.

Deprecated resource. Support for the resource might be removed in a later release.

Configuration for a vertical pod autoscaler, which automatically manages pod resources based on historical and real time resource utilization.

- **apiVersion**
  string

  Defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values.

  More information in the documentation.

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase.

  More information in the documentation.

- **metadata**
  object

- **spec**
  object

  Required value

  Specification of the behavior of the autoscaler.

  More information in the documentation.

  - **spec.recommenders**
    array of objects

    Recommender responsible for generating recommendation for this object. List should be empty (then the default recommender will generate the recommendation) or contain exactly one recommender.

    VerticalPodAutoscalerRecommenderSelector points to a specific Vertical Pod Autoscaler recommender. In the future it might pass parameters to the recommender.

    - **spec.recommenders.name**
      string

      Required value

      Name of the recommender responsible for generating recommendation for this object.

  - **spec.resourcePolicy**
    object

    Controls how the autoscaler computes recommended resources. The resource policy may be used to set constraints on the recommendations for individual containers.

    If any individual containers need to be excluded from getting the VPA recommendations, then it must be disabled explicitly by setting mode to `Off` under `containerPolicies`.

    If not specified, the autoscaler computes recommended resources for all containers in the pod, without additional constraints.

    - **spec.resourcePolicy.containerPolicies**
      array of objects

      Per-container resource policies.

      Controls how autoscaler computes the recommended resources for a specific container.

      - **spec.resourcePolicy.containerPolicies.containerName**
        string

        Name of the container or `DefaultContainerResourcePolicy`, in which case the policy is used by the containers that don’t have their own policy specified.

      - **spec.resourcePolicy.containerPolicies.controlledResources**
        array of strings

        Specifies the type of recommendations that will be computed (and possibly applied) by VPA. If not specified, the default of (`ResourceCPU`, `ResourceMemory`) will be used.

        Default: `["cpu","memory"]`

        - **Element of the array**
          string

          ResourceName is the name identifying various resources in a ResourceList.

      - **spec.resourcePolicy.containerPolicies.controlledValues**
        string

        Specifies which resource values should be controlled.

        Default: `RequestsAndLimits`

        Allowed values: `RequestsAndLimits`, `RequestsOnly`

      - **spec.resourcePolicy.containerPolicies.maxAllowed**
        object

        Specifies the maximum amount of resources that will be recommended for the container. The default is no maximum.

        - **spec.resourcePolicy.containerPolicies.minAllowed**
          object

          Specifies the minimal amount of resources that will be recommended for the container. The default is no minimum.

        - **spec.resourcePolicy.containerPolicies.mode**
          string

          Whether autoscaler is enabled for the container.

          Default: `Auto`

          Allowed values: `Auto`, `Off`

    - **spec.scope**
      string

      Scope controls additional recommendation partitioning for DaemonSet targets.

      The value must be a node label key (for example: `node.kubernetes.io/instance-type`).

    - **spec.targetRef**
      object

      Required value

      Points to the controller managing the set of pods for the autoscaler to control — e.g. Deployment, StatefulSet.

      VerticalPodAutoscaler can be targeted at controller implementing scale subresource (the pod set is retrieved from the controller's `ScaleStatus`) or some well known controllers (e.g. for DaemonSet the pod set is read from the controller's spec).

      If VerticalPodAutoscaler cannot use specified target it will report `ConfigUnsupported` condition. Note that VerticalPodAutoscaler does not require full implementation of scale subresource — it will not use it to modify the replica count. The only thing retrieved is a label selector matching pods grouped by the target resource.

        - **spec.targetRef.apiVersion**
          string

          API version of the referent.

        - **spec.targetRef.kind**
          string

          Required value

          Kind of the referent.

          More info.

        - **spec.targetRef.name**
          string

          Required value

          Name of the referent.

          More info.

    - **spec.updatePolicy**
      object

      Describes the rules on how changes are applied to the pods. If not specified, all fields in the `PodUpdatePolicy` are set to their default values.

        - **spec.updatePolicy.evictionRequirements**
          array of objects

          EvictionRequirements is a list of EvictionRequirements that need to evaluate to true in order for a Pod to be evicted. If more than one EvictionRequirement is specified, all of them need to be fulfilled to allow eviction.

          EvictionRequirement defines a single condition which needs to be true in order to evict a Pod.

            - **spec.updatePolicy.evictionRequirements.changeRequirement**

string

Required value

EvictionChangeRequirement refers to the relationship between the new target recommendation for a Pod and its current requests, what kind of change is necessary for the Pod to be evicted.

Allowed values: `TargetHigherThanRequests`, `TargetLowerThanRequests`

- **spec.updatePolicy.evictionRequirements.resources**
  array of strings

  Required value

  Resources is a list of one or more resources that the condition applies to. If more than one resource is given, the EvictionRequirement is fulfilled if at least one resource meets `changeRequirement`.

  - **Element of the array**
    string

    ResourceName is the name identifying various resources in a ResourceList.

- **spec.updatePolicy.minReplicas**
  integer

  Minimal number of replicas which need to be alive for Updater to attempt pod eviction (pending other checks like PDB). Only positive values are allowed. Overrides global '–min-replicas' flag.

- **spec.updatePolicy.updateMode**
  string

  The operation mode of the VPA controller.

  - `Auto` — currently, Auto and Recreate modes do the same thing. This mode is reserved for the Pod inplace resource update in Kubernetes.
  - `Recreate` — this mode allows VPA to modify resource requirements of the running pods (i.e., restart them during operation). This mode might result in temporary unavailability of the service (due to restart) if you have just one replica (replicas: 1). In this mode, VPA does not recreate pods that were not created by a controller.
  - `Initial` — VPA modifies pod resources only when pods are started (but not when pods are running).
  - `Off` — VPA does not take any action to update the resource requests for the running containers. Autoscaler calculates recomendations and stores them in the status field of the VPA object. You can browse VPA recommendations using the following command: `d8 k describe vpa <vpa-name>`.
  - `InPlaceOrRecreate` (GA) — supported in Kubernetes 1.33 and later. The VPA attempts to update pod resources without a restart; if that is not possible, it recreates the pod.

  Default: `InPlaceOrRecreate`

  Allowed values: `Off`, `Initial`, `Recreate`, `Auto`, `InPlaceOrRecreate`

- **status**
  object

  Current information about the autoscaler.

  - **status.conditions**
    array of objects

    Set of conditions required for this autoscaler to scale its target, and indicates whether or not those conditions are met.

    Describes the state of a VerticalPodAutoscaler at a certain point.

    - **status.conditions.lastTransitionTime**
      string

      Last time the condition transitioned from one status to another.

    - **status.conditions.message**
      string

A human-readable explanation containing details about the transition.

- **status.conditions.reason**
  string

  Reason for the condition’s last transition.

- **status.conditions.status**
  string

  Required value

  Status of the condition (`True`, `False`, Unknown).

- **status.conditions.type**
  string

  Required value

  Describes the current condition.

- **status.groups**
  array of objects

  Contains grouped recommendations by scope value when `spec.scope` is used.

  - **status.groups.containerRecommendations**
    array of objects

    Resources recommended by autoscaler for each container in this group.

    - **status.groups.containerRecommendations.containerName**
      string

      Name of the container.

    - **status.groups.containerRecommendations.lowerBound**
      object

      Minimum recommended amount of resources. Observes `ContainerResourcePolicy`. This amount is not guaranteed to be sufficient for the application to operate in a stable way, however running with less resources is likely to have significant impact on performance/availability.

    - **status.groups.containerRecommendations.scope**
      string

      Scope binds this recommendation to a specific scope value in form “ =".

    - **status.groups.containerRecommendations.target**
      object

      Required value

      Recommended amount of resources. Observes `ContainerResourcePolicy`.

    - **status.groups.containerRecommendations.uncappedTarget**
      object

      The most recent recommended resources target computed by the autoscaler for the controlled pods, based only on actual resource usage, not taking into account the `ContainerResourcePolicy`.

      May differ from the Recommendation if the actual resource usage causes the target to violate the `ContainerResourcePolicy` (lower than MinAllowed or higher that MaxAllowed). Used only as status indication, will not affect actual resource assignment.

    - **status.groups.containerRecommendations.upperBound**
      object

      Maximum recommended amount of resources.

      Observes `ContainerResourcePolicy`. Any resources allocated beyond this value are likely wasted. This value may be larger than the maximum amount of application is actually capable of consuming.

  - **status.groups.scopeValue**

string

Required value

Scope value of `spec.scope` label key for this recommendation group.

- **status.recommendation**
  object

  The most recently computed amount of resources recommended by the autoscaler for the controlled pods.

  - **status.recommendation.containerRecommendations**
    array of objects

    Resources recommended by the autoscaler for each container.

    Recommendation of resources computed by autoscaler for a specific container. Respects the container resource policy if present in the spec. In particular the recommendation is not produced for containers with `ContainerScalingMode` set to 'Off'.

    - **status.recommendation.containerRecommendations.containerName**
      string

      Name of the container.

    - **status.recommendation.containerRecommendations.lowerBound**
      object

      Minimum recommended amount of resources. Observes `ContainerResourcePolicy`. This amount is not guaranteed to be sufficient for the application to operate in a stable way, however running with less resources is likely to have significant impact on performance/availability.

    - **status.recommendation.containerRecommendations.scope**
      string

      Scope binds this recommendation to a specific scope value in form " =".

    - **status.recommendation.containerRecommendations.target**
      object

      Required value

      Recommended amount of resources. Observes `ContainerResourcePolicy`.

    - **status.recommendation.containerRecommendations.uncappedTarget**
      object

      The most recent recommended resources target computed by the autoscaler for the controlled pods, based only on actual resource usage, not taking into account the `ContainerResourcePolicy`. May differ from the Recommendation if the actual resource usage causes the target to violate the `ContainerResourcePolicy` (lower than MinAllowed or higher that MaxAllowed). Used only as status indication, will not affect actual resource assignment.

    - **status.recommendation.containerRecommendations.upperBound**
      object

      Maximum recommended amount of resources. Observes `ContainerResourcePolicy`. Any resources allocated beyond this value are likely wasted. This value may be larger than the maximum amount of application is actually capable of consuming.

VerticalPodAutoscalerCheckpoint

Short names: vpacheckpoint

Scope: Namespaced

- v1
- v1beta2

VerticalPodAutoscalerCheckpoint is the checkpoint of the internal state of VPA that is used for recovery after recommender's restart.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More information in the documentation.

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More information in the documentation.

- **metadata**
  object
- **spec**
  object

  Specification of the checkpoint. More information in the documentation.

  - **spec.containerName**
    string

    Name of the checkpointed container.

  - **spec.vpaObjectName**
    string

    Name of the VPA object that stored VerticalPodAutoscalerCheckpoint. object.

- **status**
  object

  Data of the checkpoint.

  - **status.cpuHistogram**
    object

    Checkpoint of histogram for consumption of CPU.

    - **status.cpuHistogram.bucketWeights**
      object

      Map from bucket index to bucket weight.

    - **status.cpuHistogram.referenceTimestamp**
      string

      Reference timestamp for samples collected within this histogram.

    - **status.cpuHistogram.totalWeight**
      number

      Sum of samples to be used as denominator for weights from BucketWeights.

  - **status.firstSampleStart**
    string

    Timestamp of the fist sample from the histograms.

  - **status.lastSampleStart**
    string

    Timestamp of the last sample from the histograms.

  - **status.lastUpdateTime**
    string

    The time when the status was last refreshed.

  - **status.memoryHistogram**
    object

    Checkpoint of histogram for consumption of memory.

    - **status.memoryHistogram.bucketWeights**
      object

Map from bucket index to bucket weight.

- **status.memoryHistogram.referenceTimestamp**
  string

  Reference timestamp for samples collected within this histogram.

- **status.memoryHistogram.totalWeight**
  number

  Sum of samples to be used as denominator for weights from BucketWeights.

- **status.totalSamplesCount**
  integer

  Total number of samples in the histograms.

- **status.version**
  string

  Version of the format of the stored data.

Deprecated resource. Support for the resource might be removed in a later release.

VerticalPodAutoscalerCheckpoint is the checkpoint of the internal state of VPA that is used for recovery after recommender’s restart.

- **apiVersion**
  string

  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More information in the documentation.

- **kind**
  string

  Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More information in the documentation.

- **metadata**
  object
- **spec**
  object

  Specification of the checkpoint. More information in the documentation.

  - **spec.containerName**
    string

    Name of the checkpointed container.

  - **spec.vpaObjectName**
    string

    Name of the VPA object that stored VerticalPodAutoscalerCheckpoint. object.

- **status**
  object

  Data of the checkpoint.

  - **status.cpuHistogram**
    object

    Checkpoint of histogram for consumption of CPU.

    - **status.cpuHistogram.bucketWeights**
      object

      Map from bucket index to bucket weight.

    - **status.cpuHistogram.referenceTimestamp**
      string

      Reference timestamp for samples collected within this histogram.

    - **status.cpuHistogram.totalWeight**

number

Sum of samples to be used as denominator for weights from BucketWeights.

- **status.firstSampleStart**
  string

  Timestamp of the fist sample from the histograms.

- **status.lastSampleStart**
  string

  Timestamp of the last sample from the histograms.

- **status.lastUpdateTime**
  string

  The time when the status was last refreshed.

- **status.memoryHistogram**
  object

  Checkpoint of histogram for consumption of memory.

  - **status.memoryHistogram.bucketWeights**
    object

    Map from bucket index to bucket weight.

  - **status.memoryHistogram.referenceTimestamp**
    string

    Reference timestamp for samples collected within this histogram.

  - **status.memoryHistogram.totalWeight**
    number

    Sum of samples to be used as denominator for weights from BucketWeights.

- **status.totalSamplesCount**
  integer

  Total number of samples in the histograms.

- **status.version**
  string

  Version of the format of the stored data.

FAQ

> **i** The module lifecycle stage: General Availability
> The module has requirements for installation

How do I view the Vertical Pod Autoscaler recommendations?

You can view the VPA recommendations after the VerticalPodAutoscaler custom resource is created using the following command:

```
d8 k describe vpa my-app-vpa
```

The `status` will have the following parameters:

- `Target` — the optimal amount of resources for the Pod (within the resourcePolicy).
- `Lower Bound` — the minimum recommended amount of resources for the regular operation of the application.
- `Upper Bound` — the maximum recommended amount of resources. Most likely, the resources above this upper bound will never be used by the application.
- `Uncapped Target` — the recommended amount of resources based on the latest metrics (the history of resource usage is ignored).

How does Vertical Pod Autoscaler handle limits?

Example No. 1

The following example shows a VPA object:

```
---
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: test2
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: test2
  updatePolicy:
    updateMode: "Initial"
```

The VPA object contains a Pod with the following resources:

```
resources:
  limits:
    cpu: 2
  requests:
    cpu: 1
```

If a container uses all the CPU, and VPA recommends 1.168 CPU for that container, then the ratio between requests and limits will be 100%. In this case, when recreating the Pod, VPA will modify it and set the following resources:

```
resources:
  limits:
    cpu: 2336m
  requests:
    cpu: 1168m
```

Example No. 2

The following example shows a VPA object:

```
---
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: test2
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: test2
  updatePolicy:
    updateMode: "Initial"
```

The VPA object contains a pod with resources:

```
resources:
  limits:
    cpu: 1
  requests:
    cpu: 750m
```

If the request-to-limit ratio is 25% and VPA recommends 1.168 CPU for the container, VPA will change

the container resources as follows:

```
resources:
  limits:
    cpu: 1557m
  requests:
    cpu: 1168m
```

If you need to limit the maximum number of resources that can be allocated to container constraints, you should use `maxAllowed` in the VPA object specification or use the Limit Range of the Kubernetes object.

Examples

> **i**
> The module lifecycle stage: General Availability
> The module has requirements for installation

The module configuration

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: vertical-pod-autoscaler
spec:
  version: 1
  enabled: true
  settings:
    nodeSelector:
      node-role/system: ""
    tolerations:
    - key: dedicated.deckhouse.io
      operator: Equal
      value: system
```

The basic `VerticalPodAutoscaler` custom resource example

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-app-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: StatefulSet
    name: my-app
```

The advanced `VerticalPodAutoscaler` custom resource example

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-app-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: my-app
  updatePolicy:
    updateMode: "InPlaceOrRecreate"
  resourcePolicy:
    containerPolicies:
    - containerName: hamster
      minAllowed:
        memory: 100Mi
        cpu: 120m
      maxAllowed:
        memory: 300Mi
        cpu: 350m
      mode: Auto
```