

# **Deckhouse Kubernetes Platform User's guide**

Version v1.76

Generated: September 01, 2026

## Contents

|                                                                                         |    |
|-----------------------------------------------------------------------------------------|----|
| Contents                                                                                | 2  |
| Configuring authentication for applications                                             | 7  |
| Overview                                                                                | 7  |
| Interface                                                                               | 7  |
| Changing and resetting a local user's password                                          | 8  |
| Password change required by policy or administrator                                     | 8  |
| Self-service password reset                                                             | 8  |
| Enabling Authentication in a Web Application                                            | 8  |
| Authentication via proxy (for applications without OIDC support)                        | 8  |
| Authentication for applications with OIDC support                                       | 10 |
| Additional resources                                                                    | 11 |
| Configuring Vertical Pod Autoscaler (VPA) for Applications                              | 11 |
| How Vertical Pod Autoscaler (VPA) works                                                 | 11 |
| How VPA interacts with limits                                                           | 11 |
| Monitoring VPA in Grafana                                                               | 13 |
| VPA configuration examples                                                              | 13 |
| Incoming traffic processing                                                             | 14 |
| NLB features and purpose                                                                | 14 |
| ALB features and purpose                                                                | 14 |
| Utilizing Network Load Balancer (NLB)                                                   | 15 |
| Examples of Service configuration                                                       | 15 |
| Shared IP address for multiple Services                                                 | 15 |
| Forcing an IP address assignment                                                        | 15 |
| Assigning an IPAddressPool (BGP mode)                                                   | 16 |
| Utilizing Application Load Balancer (ALB)                                               | 16 |
| Recommendations for choosing and features of different types of ALBs                    | 16 |
| Ingress-nginx                                                                           | 16 |
| Kubernetes Gateway API                                                                  | 16 |
| Istio                                                                                   | 17 |
| Publishing applications using the Ingress NGINX Controller                              | 17 |
| Publishing applications using the Kubernetes Gateway API                                | 17 |
| Publishing an application through a ClusterALBInstance object                           | 17 |
| Publishing an application through a ALBInstance object                                  | 19 |
| Working with GRPCRoute, TLSRoute, and TCPRoute objects                                  | 20 |
| Publishing the app via a different gateway                                              | 24 |
| Linking routes in one namespace to ListenerSet object in another                        | 24 |
| Configuring TLS parameters with BackendTLSPolicy                                        | 25 |
| Supported HTTPRoute annotations                                                         | 27 |
| Publishing applications using Istio                                                     | 27 |
| Publishing applications using NGINX Ingress                                             | 28 |
| Publishing applications using Istio Ingress Gateway resource                            | 28 |
| gRPC load balancing                                                                     | 29 |
| Additional resources                                                                    | 29 |
| Additional networks for use in application pods                                         | 29 |
| Creating a project network (user network)                                               | 30 |
| Connecting additional networks to pods                                                  | 30 |
| IPAM for additional networks (IP allocation and assignment)                             | 31 |
| Features of using IPAM in a DKP cluster                                                 | 31 |
| Allocating a pool of IP addresses for the project network and enabling IPAM             | 31 |
| Assigning IP addresses to network interfaces of pods connected to an additional network | 32 |
| Automatic IP address assignment                                                         | 32 |
| Manual (explicit) creation of IPAddress with type Auto                                  | 33 |
| Manually assigning a static IP address to an additional pod interface                   | 34 |
| Verification of IP address assignment to an Interface                                   | 35 |
| Additional resources                                                                    | 35 |
| Connecting physical network interfaces to DPDK application pods                         | 36 |
| Connecting physical network interfaces to pods                                          | 36 |
| Creating a pod with a device from the Underlay network                                  | 36 |
| Intra-cluster communication                                                             | 39 |
| Standard load balancer                                                                  | 39 |

|                                                                          |    |
|--------------------------------------------------------------------------|----|
| Limitations of the standard Service load balancer                        | 40 |
| Example Service for a standard load balancer                             | 40 |
| Example of a standard load balancer for a PostgreSQL cluster             | 40 |
| Creating a StatefulSet for PostgreSQL                                    | 40 |
| Advanced load balancer                                                   | 41 |
| Load balancer structure                                                  | 41 |
| Configuring the load balancer                                            | 41 |
| Example configuration of advanced ServiceWithHealthchecks load balancers | 42 |
| Hosting two independent load balancers on a single virtual machine       | 43 |
| Creating the virtual machine                                             | 43 |
| Manifests for advanced load balancers for the web and SMTP services      | 44 |
| Additional resources                                                     | 45 |
| Managing authorization and access to workload with Istio                 | 45 |
| Authorization                                                            | 45 |
| AuthorizationPolicy resource                                             | 45 |
| Additional resources                                                     | 46 |
| Configuring request routing with Istio                                   | 46 |
| VirtualService resource                                                  | 46 |
| Additional resources                                                     | 46 |
| Enabling Istio for applications                                          | 46 |
| Additional resources                                                     | 47 |
| Managing request balancing between service endpoints with Istio          | 47 |
| DestinationRule resource                                                 | 47 |
| Additional resources                                                     | 47 |
| Configuring Locality failover with Istio                                 | 47 |
| Additional resources                                                     | 48 |
| Configuring request retries with Istio                                   | 48 |
| VirtualService resource                                                  | 48 |
| Additional resources                                                     | 49 |
| Configuring resources for istio-proxy sidecars                           | 49 |
| Supported annotations                                                    | 49 |
| Configuration examples                                                   | 49 |
| Additional resources                                                     | 50 |
| Configuring Circuit Breaker                                              | 50 |
| Example Circuit Breaker configuration                                    | 50 |
| Additional resources                                                     | 51 |
| Configuring Canary deployment                                            | 51 |
| Example Canary deployment configuration with Ingress NGINX               | 51 |
| Creating a Deployment and Service for the stable version                 | 51 |
| Creating a Deployment and Service for the Canary version                 | 52 |
| Configuring Ingress for Canary deployment                                | 53 |
| Gradually increasing traffic to the Canary version                       | 54 |
| Rolling back or completing the Canary deployment                         | 54 |
| Additional annotations for Canary deployment                             | 54 |
| Example Canary deployment configuration with Istio                       | 55 |
| Distribution based on the presence of a cookie                           | 55 |
| Distribution based on probability                                        | 56 |
| Additional resources                                                     | 56 |
| Configuration of container image vulnerability scanning                  | 56 |
| Accessing scan results                                                   | 56 |
| Viewing a scan report for your application                               | 56 |
| Viewing CIS compliance check results                                     | 57 |
| Viewing scan results                                                     | 57 |
| Resource types for scan results                                          | 58 |
| Resource-level security                                                  | 58 |
| VulnerabilityReport                                                      | 58 |
| ConfigAuditReport                                                        | 60 |
| ExposedSecretReport                                                      | 61 |
| SbomReport                                                               | 62 |
| Cluster-level security                                                   | 65 |
| RbacAssessmentReport                                                     | 65 |
| ClusterComplianceReport                                                  | 68 |

|                                                            |     |
|------------------------------------------------------------|-----|
| Additional resources                                       | 69  |
| Configuration of using TLS certificates                    | 69  |
| Working with certificates                                  | 69  |
| Viewing certificate details                                | 69  |
| Automatic certificate requesting                           | 70  |
| Requesting a wildcard certificate using DNS in Cloudflare  | 70  |
| Requesting a wildcard certificate using DNS in AWS Route53 | 71  |
| Requesting a wildcard certificate using DNS in Google      | 72  |
| Requesting a self-signed certificate                       | 73  |
| Generating a self-signed certificate                       | 73  |
| Certificate generation example                             | 74  |
| Protecting credentials                                     | 75  |
| Support for tls-acme annotation                            | 76  |
| Additional resources                                       | 77  |
| Deckhouse Kubernetes Platform web UI                       | 78  |
| Accessing the web UI                                       | 78  |
| Deckhouse                                                  | 78  |
| “Overview” subsection                                      | 78  |
| “Updates” subsection                                       | 79  |
| “Modules” subsection                                       | 79  |
| “Global settings” subsection                               | 80  |
| Kubernetes                                                 | 80  |
| Node management                                            | 81  |
| “Node groups” subsection                                   | 81  |
| “Instance classes” subsection                              | 86  |
| “Nodes” subsection                                         | 88  |
| “Static instances” subsection                              | 89  |
| Multitenancy                                               | 90  |
| “Projects” subsection                                      | 90  |
| “Project templates” subsection                             | 91  |
| “User namespaces” subsection                               | 93  |
| Access                                                     | 94  |
| “User operations” subsection                               | 94  |
| Network                                                    | 96  |
| “Ingress controllers” subsection                           | 96  |
| Storage                                                    | 97  |
| “Storage classes” subsection                               | 97  |
| “Persistent volume” subsection                             | 97  |
| “Local volumes” subsection                                 | 98  |
| Security                                                   | 100 |
| “CVE scanner” subsection                                   | 100 |
| Monitoring                                                 | 100 |
| “Overview” subsection                                      | 100 |
| “Metric processing” subsection                             | 101 |
| “Metric delivery” subsection                               | 102 |
| “Grafana data sources” subsection                          | 102 |
| “Grafana dashboards” subsection                            | 103 |
| “Active alerts” subsection                                 | 104 |
| Logging                                                    | 104 |
| “Log delivery” subsection                                  | 104 |
| “Log collection” subsection                                | 105 |
| Viewing API information                                    | 106 |
| Additional resources                                       | 107 |
| Grafana web interface                                      | 107 |
| Monitoring system web UI                                   | 107 |
| Main page                                                  | 107 |
| Working with dashboards                                    | 108 |
| Data filtering                                             | 109 |
| Working with data                                          | 110 |
| Dashboard descriptions                                     | 112 |
| Viewing security event logs                                | 126 |
| Additional resources                                       | 128 |



|                                                                                   |     |
|-----------------------------------------------------------------------------------|-----|
| Web interface for managing alerts                                                 | 128 |
| Accessing the web UI                                                              | 129 |
| Web interface for generating kubeconfig                                           | 130 |
| Accessing the web UI                                                              | 130 |
| Web interface for downloading CLI utilities                                       | 132 |
| Accessing the web interface                                                       | 132 |
| Additional resources                                                              | 132 |
| Secret management web interface                                                   | 132 |
| Accessing the stronghold web UI                                                   | 133 |
| Managing secrets engines                                                          | 133 |
| Viewing a secrets engine                                                          | 133 |
| Managing access to data and stronghold features                                   | 137 |
| Working with authentication methods                                               | 138 |
| User groups                                                                       | 138 |
| Entities and aliases                                                              | 139 |
| Managing temporary access to secrets and resources (leases)                       | 140 |
| Managing access control policies                                                  | 141 |
| Using additional tools                                                            | 142 |
| Monitoring stronghold Raft cluster                                                | 144 |
| Monitoring activity and load                                                      | 144 |
| Sealing and unsealing the secrets storage                                         | 145 |
| Using the stronghold CLI                                                          | 145 |
| Additional resources                                                              | 146 |
| Web interface for visualizing the cluster network stack                           | 146 |
| Accessing the web interface                                                       | 146 |
| Filtering data for display                                                        | 147 |
| Working with the network flow diagram                                             | 148 |
| Working with the network flows and events table                                   | 149 |
| Configuring the column set                                                        | 149 |
| Viewing row details                                                               | 149 |
| Using multitenancy                                                                | 150 |
| Limitations                                                                       | 150 |
| Creating a project                                                                | 150 |
| Automatically creating a project from a namespace                                 | 151 |
| Additional resources                                                              | 152 |
| Application and Infrastructure Monitoring                                         | 152 |
| Available by default                                                              | 152 |
| Dashboards                                                                        | 152 |
| Alerts                                                                            | 152 |
| Monitoring configuration                                                          | 153 |
| Additional resources                                                              | 153 |
| Monitoring dashboards                                                             | 153 |
| Dashboard types                                                                   | 153 |
| Preinstalled dashboards                                                           | 153 |
| Ingress Nginx                                                                     | 154 |
| Resource consumption (Main)                                                       | 154 |
| Security                                                                          | 154 |
| Custom dashboards                                                                 | 154 |
| Using the observability module                                                    | 154 |
| Access control                                                                    | 155 |
| Converting dashboards from GrafanaDashboardDefinition                             | 156 |
| Using GrafanaDashboardDefinition                                                  | 157 |
| Additional resources                                                              | 158 |
| Configuring Application Monitoring                                                | 158 |
| Configuring metrics collection via labels and annotations                         | 159 |
| Example of configuring metrics collection from a Service                          | 160 |
| Example of configuring metrics collection from a Deployment                       | 160 |
| Additional annotations for advanced configuration                                 | 161 |
| Configuring metrics collection using PodMonitor or ServiceMonitor resources       | 161 |
| Adding custom labels to metrics                                                   | 162 |
| Configuring metrics collection via scrape_configs using the ScrapeConfig resource | 163 |
| Configuring metrics collection using blackbox-exporter                            | 164 |

|                                                                                              |     |
|----------------------------------------------------------------------------------------------|-----|
| Additional resources                                                                         | 165 |
| Application delivery with Argo CD                                                            | 165 |
| Prerequisites                                                                                | 165 |
| AppProject projects                                                                          | 165 |
| Preparing a namespace                                                                        | 166 |
| Deploying an application                                                                     | 166 |
| Creating an application with an Application object                                           | 166 |
| Creating an application through the Argo CD web interface                                    | 167 |
| Creating an application with the argocd CLI utility                                          | 167 |
| Additional resources                                                                         | 168 |
| Marketplace                                                                                  | 168 |
| Prerequisites                                                                                | 168 |
| Additional resources                                                                         | 169 |
| Installing and managing applications                                                         | 169 |
| Browsing available package versions                                                          | 169 |
| Installing an application                                                                    | 169 |
| Naming constraints                                                                           | 170 |
| Checking application status                                                                  | 170 |
| Conditions                                                                                   | 170 |
| Summary                                                                                      | 171 |
| Multiple instances                                                                           | 171 |
| Updating an application                                                                      | 172 |
| Deleting an application                                                                      | 172 |
| FAQ                                                                                          | 172 |
| Can updates happen automatically?                                                            | 172 |
| Can an Application depend on another Application?                                            | 172 |
| Can I install the same application in different namespaces?                                  | 172 |
| Additional resources                                                                         | 172 |
| Troubleshooting                                                                              | 172 |
| Verify that Marketplace CRDs are present                                                     | 172 |
| Read the application summary                                                                 | 173 |
| Read individual conditions                                                                   | 173 |
| Check the DKP controller logs                                                                | 174 |
| Check application pod logs                                                                   | 174 |
| Common conditions and their meaning                                                          | 175 |
| Additional resources                                                                         | 175 |
| Configuration of applications logs collection                                                | 175 |
| Configuring application log collection                                                       | 175 |
| Examples                                                                                     | 176 |
| Creating a source in a namespace and reading logs from all pods in it with direction to Loki | 176 |
| Reading pods in a specified namespace with a specific label                                  | 177 |
| Additional resources                                                                         | 177 |

# Configuring authentication for applications

## Overview

Authentication is the process of verifying a user's identity. In Deckhouse Kubernetes Platform (DKP), end-to-end authentication is implemented, allowing user verification when accessing any DKP interface or cluster resources. Cluster users can also use DKP to enable authentication for their applications.

Depending on the DKP configuration, authentication can use either an internal user database or external authentication providers. Connecting an external provider allows the use of existing credentials (e.g., LDAP, GitLab, GitHub, etc.) for access. It also enables using the same credentials to authenticate in multiple DKP clusters.

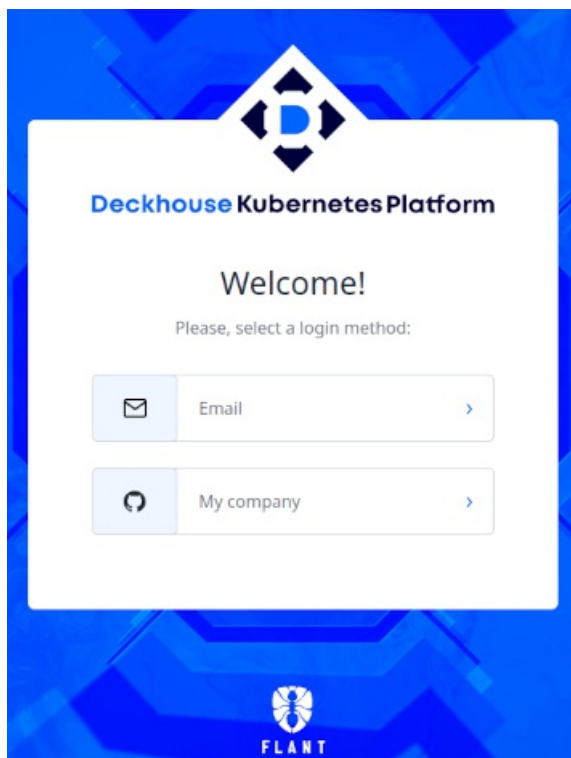
From the perspective of a cluster user or an application developer, it does not matter how the DKP administrator configured authentication — the user interface and authentication methods for applications will be the same.

**i** To use authentication in DKP, configuration is required.

## Interface

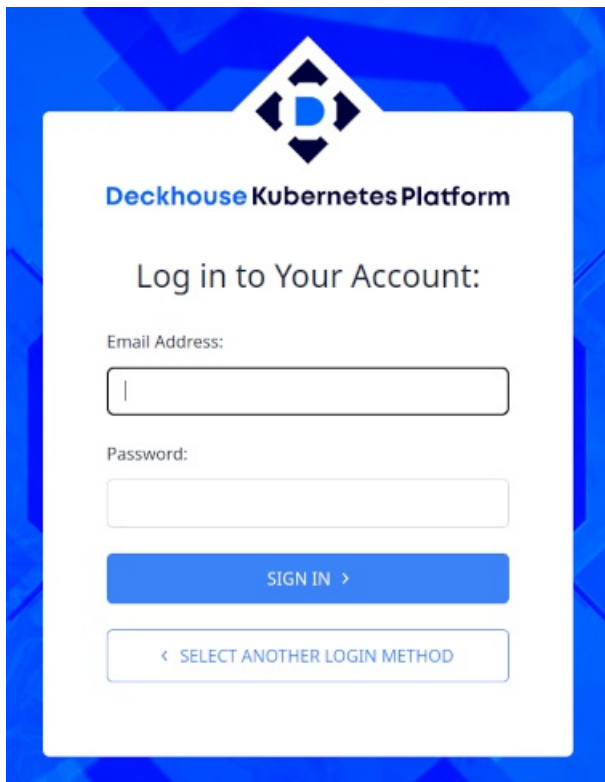
The authentication interface appears upon the first request to a resource for which authentication is enabled — DKP redirects the user to the authentication page. If the user is already authenticated (e.g., via an external identity provider), DKP will redirect the request back to the original resource, enriched with authentication data. If authentication has not yet occurred, the user will see the authentication interface.

Example of DKP authentication interface:



The authentication interface allows users to choose an authentication method if multiple methods are configured. If only one external provider is configured, the user is redirected directly to that provider's login page. If local users are defined in DKP, the user will be prompted to enter a username and password.

Example of DKP authentication interface with username and password input:



## Changing and resetting a local user's password

If local authentication is configured in the cluster, a user can change or reset their password on their own.

### Password change required by policy or administrator

During login, DKP may redirect a local user to the password change form if:

- The password complexity does not meet the configured policy.
- The password has expired according to the configured policy.
- An administrator initiated a forced password change.

The form requires the current password, a new password, and confirmation. The new password must comply with the cluster password policy.

### Self-service password reset

A local user can reset their password in the DKP authentication interface. The operation is performed via the `UserOperation` resource with `initiatorType: self`.

Self-service reset is available only for local accounts. If you sign in through LDAP, GitHub, or another external provider, contact the administrator of that system.

After a password reset, active sessions are terminated and you must sign in again with the new password.

## Enabling Authentication in a Web Application

**i** To enable authentication in an application, authentication must first be configured at the Deckhouse Kubernetes Platform level.

DKP supports two ways of enabling authentication for an application, depending on whether the application is capable of handling authentication requests (i.e., acting as an OIDC client) or not. Both approaches are described below.

### Authentication via proxy (for applications without OIDC support)

Authentication for applications that cannot independently handle authentication requests is implemented via a special proxy server. This proxy handles both authentication and authorization, hiding the details of these processes from the application.

To enable authentication for an application deployed in DKP, follow these steps:

1. Create a DexAuthenticator object in the application's namespace.

After the DexAuthenticator object is created, DKP will automatically deploy a set of components required for authentication:

- A Deployment with containers running the authentication/authorization proxy and a Redis data store;
- A Service pointing to the authentication/authorization proxy;
- An Ingress resource that handles requests at `https://<applicationDomain>/dex-authenticator` and forwards them to the proxy Service;
- Secrets required for integration with the DKP authentication system.

Example DexAuthenticator manifest:

```
apiVersion: deckhouse.io/v1
kind: DexAuthenticator
metadata:
  # Prefix for Dex authenticator pod names.
  # For example, if the name prefix is `app-name`, pods will be named like `app-
  # name-dex-authenticator-7f698684c8-c5cjg`.
  name: app-name
  # Namespace where Dex authenticator will be deployed.
  namespace: app-ns
spec:
  # Domain name of your application. Requests to this domain will be redirected
  # to Dex for authentication.
  applicationDomain: "app-name.kube.my-domain.com"
  # Whether to send an `Authorization: Bearer` header to the application. Useful
  # with NGINX's auth_request.
  sendAuthorizationHeader: false
  # Name of the Secret containing the TLS certificate.
  applicationIngressCertificateSecretName: "ingress-tls"
  # Name of the Ingress class to use for the Ingress resource created for the
  # Dex authenticator.
  applicationIngressClassName: "nginx"
  # Duration for which the user session remains active.
  keepUsersLoggedInFor: "720h"
  # List of groups whose users are allowed to authenticate.
  allowedGroups:
    - everyone
    - admins
  # List of IP addresses and CIDRs allowed to authenticate.
  whitelistSourceRanges:
    - 1.1.1.1/32
    - 192.168.0.0/24
```

Note the following capabilities when configuring authentication:

- The `applicationDomain` parameter in the DexAuthenticator defines the main domain of the application. Additional domains can be specified via the `additionalApplications.domain` parameter.
- The `whitelistSourceRanges` and `additionalApplications.whitelistSourceRanges` parameters allow you to restrict authentication access to a specific list of IP addresses.

Refer to the Authorization section for information about configuring authorization.

2. 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://<NAME>-dex-authenticator.<NS>.svc.<CLUSTER_DOMAIN>/dex-authenticator/auth`

Where:

- `<NAME>` is the value of `metadata.name` from the `DexAuthenticator`;
- `<NS>` is the value of `metadata.namespace` from the `DexAuthenticator`;
- `<CLUSTER_DOMAIN>` is the cluster domain (defined in the `clusterDomain` field of the `ClusterConfiguration`, by default it is `cluster.local`).

Example (for a `DexAuthenticator` named `app-name` in namespace `app-ns`):

```
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
```

## Authentication for applications with OIDC support

Applications that can handle authentication requests on their own and act as OIDC clients can directly integrate with the DKP authentication system. In this case, the application independently redirects the user to the login page and processes the received OIDC tokens.

To enable authentication for such an application, follow these steps:

1. Create a `DexClient` object in the application's namespace.

After the `DexClient` object is created, Deckhouse will perform the following:

- An OIDC client will be registered in the DKP authentication system with an identifier (`clientID`) in the format: `dex-client-<NAME>@<NAMESPACE>` where `<NAME>` and `<NAMESPACE>` are `metadata.name` and `metadata.namespace` from the `DexClient` resource;
  - A `clientSecret` will be automatically generated and saved in a Kubernetes Secret named `dex-client-<NAME>` in the same namespace;
  - You can then use the generated `clientID` and `clientSecret` to configure OIDC support in your application.
2. Specify the allowed redirect URIs.  
These URIs define where the provider (Dex) can redirect the user after successful authentication.
  3. Restrict access by groups, if needed.  
Use the `allowedGroups` parameter to specify which user groups are allowed to authenticate via this client.
  4. (Optional) Define trusted peers.  
Use the `trustedPeers` parameter to allow authentication delegation between applications.

Example of a `DexClient` object:

```
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
```

5. Retrieve the `clientSecret`.  
The secret will be automatically created:

```

apiVersion: v1
kind: Secret
metadata:
  name: dex-client-myname
  namespace: mynamespace
type: Opaque
data:
  clientSecret: c2VjcmV0

```

6. Configure your application as an OIDC client. Use the `clientId`, `clientSecret`, and `redirectURIs`, along with the Dex provider URL. You can get the Dex URL (`https://dex.<publicDomainTemplate>`) using the following command:

```
d8 k -n d8-user-authn get ingress dex -o jsonpath="{.spec.rules[*].host}"
```

## Additional resources

- Global custom resources
- Module `user-authn` custom resources

# Configuring Vertical Pod Autoscaler (VPA) for Applications

## How Vertical Pod Autoscaler (VPA) works

The Vertical Pod Autoscaler (VPA) automates container resource management and can significantly improve application performance. VPA is especially useful when it's difficult to estimate resource needs in advance. When VPA is used in an appropriate operating mode, it sets requested resources based on actual usage data collected from the monitoring system. You can also configure it to only provide recommendations without applying changes automatically.

## How VPA interacts with limits

VPA manages the container's resource **requests**, but it does not manage **limits** unless explicitly configured to do so.

VPA calculates recommended values based on resource usage by the container. This behavior can affect the ratio between requests and limits:

- If requests and limits are equal, VPA will only update the requests, leaving limits unchanged.
- If limits are not specified, VPA will only update requests.
- If limits are set but not controlled by VPA, the ratio between requests and limits may shift.

1. Example 1. In the cluster, we have:

- 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"

```

- A pod with specified resources:

```
resources:
limits:
  cpu: 2
requests:
  cpu: 1
```

If the container consumes 1 CPU, VPA will recommend 1.168 CPU. In this case, the ratio between requests and limits is 100% (since the request is managed, but the limit remains unchanged). When the pod is recreated, VPA will update the resources as follows:

```
resources:
limits:
  cpu: 2336m
requests:
  cpu: 1168m
```

## 2. Example 2. The cluster contains:

- 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"
```

- A pod with specified resources:

```
resources:
limits:
  cpu: 1
requests:
  cpu: 750m
```

In this case, the ratio between requests and limits will be 25%. If VPA recommends 1.168 CPU, the container's resources will be adjusted to:

```
resources:
limits:
  cpu: 1557m
requests:
  cpu: 1168m
```

If resources are not limited, VPA might assign excessively high resource values, which can cause issues.

To prevent this, you can:

- Use the `maxAllowed` parameter in the VPA specification:



```

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

```

- Configure a `LimitRange` in the cluster:

```

apiVersion: v1
kind: LimitRange
metadata:
  name: my-app-limits
spec:
  limits:
    - default:
        cpu: 2
        memory: 4Gi
      defaultRequest:
        cpu: 500m
        memory: 256Mi
      type: Container

```

## Monitoring VPA in Grafana

To efficiently manage resources using the Vertical Pod Autoscaler (VPA), it is recommended to use Grafana dashboards. These dashboards allow you to track the current status of VPA, its configuration, and the percentage of pods on which it is active.

Grafana provides several levels of detail for VPA-related information. Key dashboards include:

- Main / Namespace — Displays general VPA usage per namespace.
- Main / Namespace / Controller — Provides VPA metrics for specific controllers.
- Main / Namespace / Controller / Pod — The most granular view, showing data for each individual pod.

Key columns to monitor:

- VPA type — Shows the current value of `updatePolicy.updateMode`, which defines the VPA operating mode. This field appears in the following dashboards:
  - Main / Namespace
  - Main / Namespace / Controller
  - Main / Namespace / Controller / Pod
- VPA % (Percentage of pods with VPA enabled) — Displays the percentage of pods within a namespace that have VPA enabled. This helps quickly assess how much of the cluster is covered by automatic resource scaling via VPA.

## VPA configuration examples

1. Example of a minimal `VerticalPodAutoscaler` resource:

```

apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-app-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: StatefulSet
    name: my-app

```

## 2. Example of a full VerticalPodAutoscaler resource:

```

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

```

## Incoming traffic processing

Network Load Balancer (NLB) and Application Load Balancer (ALB) are used to provide external access to applications deployed in a cluster managed by Deckhouse Kubernetes Platform.

### NLB features and purpose

NLB operates at the transport layer. It balances TCP and UDP traffic at the IP and port level.

Key advantages:

- High performance
- Minimal latency during traffic transmission
- Simple configuration

NLB is suitable for applications that use TCP/UDP protocols, such as databases.

### ALB features and purpose

ALB operates at the application layer. It analyzes the contents of incoming requests (for example, HTTP headers, URL paths, cookies) and can route traffic based on them.

Advantages of ALB:

- Support for HTTP(S) protocols and gRPC
- Flexible routing (path-based, host-based)
- Ability to terminate SSL/TLS
- Integration with authentication and authorization mechanisms

ALB is suitable for web applications, APIs, and other services where intelligent routing and working with HTTP requests are important.

## Utilizing Network Load Balancer (NLB)

NLB is provided by using Services of LoadBalancer type.

### Examples of Service configuration

#### Shared IP address for multiple Services

To make Services use the same IP address, add the `network.deckhouse.io/load-balancer-shared-ip-key` annotation to them:

```
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
```

#### Forcing an IP address assignment

To enforce an IP address for a Service, add the `network.deckhouse.io/load-balancer-ips` annotation:

```

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

```

## Assigning an IPAddressPool (BGP mode)

In BGP LoadBalancer mode, an IP address can be allocated from a specific address pool using the `metallb.universe.tf/address-pool` annotation. For L2 LoadBalancer mode, you need to use the `MetalLoadBalancerClass` configuration.

Example:

```

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

```

## Utilizing Application Load Balancer (ALB)

Application deployment and application-level traffic balancing can be performed using the following tools:

- Ingress NGINX Controller (`ingress-nginx` module).
- Kubernetes Gateway API (`alb` module).
- Istio (`istio` module).

## Recommendations for choosing and features of different types of ALBs

### Ingress-nginx

ALB, powered by Ingress NGINX Controller, is based on the nginx web server and is implemented by a `ingress-nginx` module. This option is suitable for:

- Basic traffic routing based on domains or URLs.
- Using SSL/TLS to secure traffic.

### Kubernetes Gateway API

ALB is implemented using the Kubernetes Gateway API via the `alb` module. Gateways run on Envoy Proxy, and reception and routing are described using standard API objects (Gateway, ListenerSet, HTTPRoute, and, if necessary, GRPCRoute, TLSRoute, TCPRoute, BackendTLSPolicy). The controller deploys the necessary ingress infrastructure and validates the configuration to prevent conflicting handlers.

You should choose this option if you need:

- To publish applications using the Gateway API model instead of the classic Ingress.
- A cluster-wide entry point or a separate gateway for an application or team within your namespace.
- HTTP/HTTPS and gRPC routing, TLS termination or pass-through, as well as TCP after TLS termination at the gateway.
- Route parameters not included in the specification, via `HTTPRoute` annotations.

## Istio

An ALB based on `istio` module provides advanced traffic management capabilities. Consider an istio-based ALB if you need:

- Advanced routing, for example, to implement canary deployment.
- Traffic distribution between application versions and microservices.
- mTLS for encrypting traffic between Pods.
- Request tracing.

## Publishing applications using the Ingress NGINX Controller

To publish applications, the cluster administrator must create an Ingress controller. Specify the name of this object in the Ingress resource manifest, which is used to route incoming traffic to your application.

Example of a basic Ingress resource for publishing an application.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  ingressClassName: nginx # The name of the Ingress controller provided by the
cluster administrator.
  rules:
  - host: application.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: productpage
            port:
              number: 80
```

## Publishing applications using the Kubernetes Gateway API

Applications can be deployed via a cluster-wide gateway (using a `ClusterALBInstance` object created by the cluster administrator) or via a separate gateway for an application or pod in a dedicated namespace (using a `ClusterALBInstance` object).

### Publishing an application through a `ClusterALBInstance` object

This scenario assumes that the `ClusterALBInstance` object has already been created by an administrator and has reached the `Ready` state. The name and namespace of the managed Gateway object should be taken from the `status` of the `ClusterALBInstance` object.

Next, create a `ListenerSet` object that will be bound to the desired gateway (using the `spec.parentRef.name` parameter) and `HTTPRoute` objects (routes) to route incoming requests to the application. Example:

```
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
---
# 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 # HTTP traffic always uses 80 regardless of ClusterALBInstance
settings.
    hostnames:
      - app.example.com
    rules:
      - backendRefs:
          - name: app-svc # Reference to the internal load balancer of the application.
            port: 8080
---
# Route for HTTP 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 an application through a ALBInstance object

In this scenario, the ALBInstance object, the Gateway object, the ListenerSet object, and the HTTPRoute object live in the same namespace.

To publish an application using the ALBInstance object, follow these steps:

1. Create the ALBInstance object taking into account the required settings:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: app-gw
  namespace: prod
spec:
  gatewayName: app-gw
  inlet:
    type: LoadBalancer
```

2. After the ALBInstance object reaches the **Ready** state, create the ListenerSet object and the HTTPRoute object:

```

apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: app-listeners
  namespace: prod
spec:
  parentRef:
    name: app-gw # The name of the Gateway object from the ClusterALBInstance.
    namespace: prod
  listeners:
    - name: app-https
      port: 443
      protocol: HTTPS
      hostname: app.example.com
      tls:
        mode: Terminate
        certificateRefs:
          - name: app-tls # Reference to the Secret with the TLS certificate.
            namespace: prod
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: app-route
  namespace: prod
spec:
  parentRefs:
    - name: app-listeners # ListenerSet name.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: app-https
      port: 443
  hostnames:
    - app.example.com
  rules:
    - backendRefs:
        - name: app-svc # Reference to the internal load balancer of the
          application.
            port: 8080

```

## Working with GRPCRoute, TLSRoute, and TCPRoute objects

The GRPCRoute object is intended for gRPC traffic. For it, create the ListenerSet object with an HTTPS listener, then add the GRPCRoute object:



```

apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: grpc-listeners
  namespace: prod
spec:
  parentRef:
    name: app-gw # The name of the Gateway object from the ClusterALBInstance.
    namespace: prod
  listeners:
    - name: grpc-https
      port: 443
      protocol: HTTPS
      hostname: grpc.example.com
      tls:
        mode: Terminate
        certificateRefs:
          - name: grpc-tls # Reference to the Secret with the TLS certificate.
            namespace: prod
---
apiVersion: gateway.networking.k8s.io/v1
kind: GRPCRoute
metadata:
  name: grpc-route
  namespace: prod
spec:
  parentRefs:
    - name: grpc-listeners # ListenerSet name.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: grpc-https
      port: 443
  hostnames:
    - grpc.example.com
  rules:
    - backendRefs:
        - name: grpc-svc # Reference to the Secret with the TLS certificate.
          port: 9090

```

For TLS passthrough, when traffic must be decrypted on the application side, either a TLS listener or an HTTPS listener can be used. The example below shows the TLS listener variant:

To accept TCP traffic on an additional port, configure the `additionalPorts` parameter in the `ALBInstance`:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: app-gw
  namespace: prod
spec:
  gatewayName: app-gw
  inlet:
    type: LoadBalancer
    additionalPorts:
      - port: 8443 # An additional TCP port to accept TLS traffic.
        protocol: TCP

```

Next, configure `ListenerSet` and `TLSRoute` objects respectively:

```

apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: tls-pass-listeners
  namespace: prod
spec:
  parentRef:
    name: app-gw # The name of the Gateway object from the ClusterALBInstance.
    namespace: prod
  listeners:
    - name: tls-pass
      port: 8443 # In this case 8443 port is used for TLS.
      protocol: TLS
      hostname: pass.example.com
      tls:
        mode: Passthrough # TLS passthrough mode is set explicitly.
---
apiVersion: gateway.networking.k8s.io/v1alpha3
kind: TLSRoute
metadata:
  name: tls-pass-route
  namespace: prod
spec:
  parentRefs:
    - name: tls-pass-listeners # ListenerSet name.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: tls-pass
      port: 8443 # In this case 8443 port is used for TLS.
  hostnames:
    - pass.example.com
  rules:
    - backendRefs:
        - name: tls-pass-svc # Reference to the internal load balancer of the
          application.
            port: 8443

```

The same scenario can also be implemented through an HTTPS listener. This variant is especially useful when the standard handler on port `443` should be used because no extra port needs to be opened for TLS passthrough:

```

apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: https-pass-listeners
  namespace: prod
spec:
  parentRef:
    name: app-gw # The name of the Gateway object from the ClusterALBInstance.
    namespace: prod
  listeners:
    - name: https-pass
      port: 443 # In this case 443 (HTTPS) port is reused for TLS.
      protocol: HTTPS
      hostname: pass.example.com
      tls:
        mode: Passthrough # TLS passthrough mode is set explicitly.
---
apiVersion: gateway.networking.k8s.io/v1alpha3
kind: TLSRoute
metadata:
  name: https-pass-route
  namespace: prod
spec:
  parentRefs:
    - name: https-pass-listeners # ListenerSet name.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: https-pass
      port: 443 # In this case 443 (HTTPS) port is reused for TLS.
  hostnames:
    - pass.example.com
  rules:
    - backendRefs:
        - name: tls-pass-svc # Reference to the internal load balancer of the
          application.
            port: 8443

```

If TLS must be terminated on the gateway and then the traffic must be passed further as a regular TCP stream, create a ListenerSet object with a TLS listener in `Terminate` mode, then attach a TCPRoute object:

```

apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: tls-term-listeners
  namespace: prod
spec:
  parentRef:
    name: app-gw # The name of the Gateway object from the ClusterALBInstance.
    namespace: prod
  listeners:
    - name: tls-term
      port: 443 # In this case 443 (HTTPS) port is reused for TLS.
      protocol: TLS
      hostname: term.example.com
      tls:
        mode: Terminate
        certificateRefs:
          - name: term-tls # Reference to the Secret with the TLS certificate.
            namespace: prod
---
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: TCPRoute
metadata:
  name: tls-term-route
  namespace: prod
spec:
  parentRefs:
    - name: tls-term-listeners # ListenerSet name.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: tls-term
      port: 443 # In this case 443 (HTTPS) port is reused for TLS.
  rules:
    - backendRefs:
        - name: tcp-svc # Reference to the internal load balancer of the application.
          port: 8080

```

## Publishing the app via a different gateway

If an application needs to move to another managed Gateway object, change the route attachment in stages:

1. Create a new ClusterALBInstance object or ALBInstance object so that the controller creates a new Gateway object.
2. Create a ListenerSet object with the same hostnames, ports, and TLS settings. The new Gateway object must be specified in `spec.parentRef`.
3. Add one more `parentRefs` entry to the existing HTTPRoute object, pointing to the new ListenerSet object.
4. Verify traffic through the new gateway path.
5. After verification, remove the reference to the obsolete ListenerSet object from `parentRefs` of the HTTPRoute object.

## Linking routes in one namespace to ListenerSet object in another

If an HTTPRoute object is created in one namespace and must be attached to a ListenerSet object in another namespace, add a ReferenceGrant object in the namespace of the target ListenerSet object. The example below shows a shared ListenerSet object in namespace `shared-gw`, an application HTTPRoute object in namespace `prod`, and a ReferenceGrant object that allows this attachment:

```

apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: shared-listeners
  namespace: shared-gw
spec:
  parentRef:
    name: public-gw
    namespace: d8-alb
  listeners:
    - name: app-https
      port: 443
      protocol: HTTPS
      hostname: app.example.com
      tls:
        mode: Terminate
        certificateRefs:
          - name: app-tls
            namespace: shared-gw
---
apiVersion: gateway.networking.k8s.io/v1
kind: ReferenceGrant
metadata:
  name: allow-prod-httproute-to-shared-listeners
  namespace: shared-gw
spec:
  from:
    - group: gateway.networking.k8s.io
      kind: HTTPRoute
      namespace: prod
  to:
    - group: gateway.networking.k8s.io
      kind: ListenerSet
      name: shared-listeners
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: app-route
  namespace: prod
spec:
  parentRefs:
    - name: shared-listeners
      namespace: shared-gw
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: app-https
      port: 443
  hostnames:
    - app.example.com
  rules:
    - backendRefs:
        - name: app-svc
          port: 8080

```

## Configuring TLS parameters with BackendTLSPolicy

If traffic from the gateway to the backend must use TLS, create a BackendTLSPolicy object in the namespace of the backend Service object. The example below shows an HTTPRoute object, a backend Service object with a named port, a ConfigMap with a CA bundle, and a BackendTLSPolicy object that configures TLS validation for that backend:

```

apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: app-route
  namespace: prod
spec:
  parentRefs:
    - name: app-listeners
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: app-https
      port: 443
  hostnames:
    - app.example.com
  rules:
    - backendRefs:
        - name: app-svc
          port: 8443
---
apiVersion: v1
kind: Service
metadata:
  name: app-svc
  namespace: prod
spec:
  selector:
    app: app
  ports:
    - name: https
      port: 8443
      targetPort: 8443
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-backend-ca
  namespace: prod
data:
  ca.crt: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
---
apiVersion: gateway.networking.k8s.io/v1
kind: BackendTLSPolicy
metadata:
  name: app-svc-tls
  namespace: prod
spec:
  targetRefs:
    - group: ""
      kind: Service
      name: app-svc
      sectionName: https
  validation:
    hostname: app.internal.example.com
    caCertificateRefs:
      - group: ""
        kind: ConfigMap
        name: app-backend-ca

```

## Supported HTTPRoute annotations

Because the current Gateway API specification does not yet cover all features required for a Deckhouse cluster to operate properly, the module provides a gradually growing set of HTTPRoute object annotations that adds the missing configuration options. The controller reads these keys from `HTTPRoute.metadata.annotations`.

| Annotation                                                     | Description                                                                                                                                                                                                                                                     |
|----------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>alb.network.deckhouse.io/tls-disable-protocol</code>     | Disables a TLS protocol version for the handler with the hostname of this route (for example value <code>http2</code> ). This may be required in rare cases when a shared certificate with several DNS names is used together with request redirection.         |
| <code>alb.network.deckhouse.io/whitelist-source-range</code>   | Expects a comma-separated list of subnets in CIDR format: an IP filter at route level; overrides the global whitelist (for example <code>10.1.1.10/32, 10.2.2.2/32</code> ).                                                                                    |
| <code>alb.network.deckhouse.io/response-headers-to-add</code>  | JSON object with additional response headers (for example <code>{"Strict-Transport-Security": "max-age=31536000; includeSubDomains"}</code> ).                                                                                                                  |
| <code>alb.network.deckhouse.io/session-affinity</code>         | JSON for cookie session affinity ( <code>mode</code> , <code>path</code> , <code>cookieName</code> , <code>ttl</code> , etc.); not every field is required (for example <code>{"mode": "cookie", "path": "/path", "cookieName": "mycookie", "ttl": 0}</code> ). |
| <code>alb.network.deckhouse.io/hash-key</code>                 | For example <code>source-ip</code> : consistent hashing for Service backends of the HTTPRoute object.                                                                                                                                                           |
| <code>alb.network.deckhouse.io/service-upstream</code>         | <code>"true"</code> : traffic to the upstream goes through the corresponding Service object instead of directly to pods.                                                                                                                                        |
| <code>alb.network.deckhouse.io/basic-auth-secret</code>        | <code>namespace/secret</code> with <code>htpasswd</code> data for HTTP basic auth on this route.                                                                                                                                                                |
| <code>alb.network.deckhouse.io/satisfy</code>                  | <code>all</code> or <code>any</code> : defines whether both checks must be satisfied (whitelist and basic-auth) or only one of them (default <code>all</code> ).                                                                                                |
| <code>alb.network.deckhouse.io/auth-url</code>                 | Defines the URL of the external authentication service.                                                                                                                                                                                                         |
| <code>alb.network.deckhouse.io/auth-signin</code>              | Defines the redirect URL for authentication when <code>401</code> is returned by external authentication.                                                                                                                                                       |
| <code>alb.network.deckhouse.io/auth-response-headers</code>    | Comma-separated list: additional headers from the auth response to pass upstream (on top of the standard allowlist).                                                                                                                                            |
| <code>alb.network.deckhouse.io/rewrite-target</code>           | Allows rewriting paths for rules with <code>RegularExpression</code> type by using regex capture groups (for example <code>/my-path/\1</code> ).                                                                                                                |
| <code>alb.network.deckhouse.io/buffer-max-request-bytes</code> | Defines the buffer size that may be used when requests are buffered (by default Envoy Proxy does not buffer requests).                                                                                                                                          |
| <code>alb.network.deckhouse.io/limit-rps</code>                | RPS limit for a route.                                                                                                                                                                                                                                          |
| <code>alb.network.deckhouse.io/backend-tls-settings</code>     | For example <code>{"mode": "SIMPLE", "insecureSkipVerify": true, "clientCertificate": "", "privateKey": "", "caCertificates": ""}</code> ; allows explicit configuration of TLS connection parameters to the upstream.                                          |

## Publishing applications using Istio

When deploying an application using Istio, you can choose one of the following options:

- Using NGINX Ingress.
- Use Istio Ingress Gateway.

## Publishing applications using NGINX Ingress

To publish an application using NGINX Ingress, the Deckhouse Kubernetes Platform administrator must configure the Ingress controller by adding an Istio sidecar to it.

To publish an application, prepare an Ingress resource that references a Service. Required annotations for the Ingress resource:

- `nginx.ingress.kubernetes.io/service-upstream: "true"`: With this annotation, the Ingress controller will send requests to the Service's ClusterIP (from the Service CIDR range) instead of sending them directly to the application's Pods. The `istio-proxy` sidecar container intercepts traffic only toward the Service CIDR range. All other requests are sent directly.
- `nginx.ingress.kubernetes.io/upstream-vhost: productpage.bookinfo.svc`: With this annotation, the sidecar can identify the application service the request is intended for.

Examples:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: productpage
  namespace: bookinfo
  annotations:
    # Enables proxying traffic to the Service's ClusterIP via nginx instead of
    # directly to Pod IPs.
    nginx.ingress.kubernetes.io/service-upstream: "true"
    # In Istio, all routing is based on the `Host:` request header.
    # This avoids the need to inform Istio about the existence of the external domain
    # `productpage.example.com`;
    # the internal domain known to Istio is used instead.
    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
```

## Publishing applications using Istio Ingress Gateway resource

To publish an application using the Istio Ingress Gateway, the Deckhouse Kubernetes Platform administrator must create an `IngressIstioController` resource.



To publish an application using the Istio Ingress Gateway resource:

1. Create a Gateway resource. In the `spec.selector` field, specify the label referencing the `ingressGatewayClass` and the secret name provided by the cluster administrator:

```
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
        # Secret resource with the certificate and 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
```

2. Define routing rules using a VirtualService that links the gateway to the service it serves:

```
```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: vs-app
  namespace: app-ns
spec:
  gateways:
    - gateway-app # The name of the Gateway resource created
                  # in the previous step.
  hosts:
    - app.example.com
  http:
    - route:
        - destination:
            host: app-svc # The
                        # name of the service to which traffic should be directed.
```

## gRPC load balancing

For automatic gRPC service load balancing to work, assign a name with the prefix or value `grpc` to the port in the corresponding Service object.

## Additional resources

- Module `alb` documentation
- Module `ingress-nginx` documentation
- Module `istio` documentation

## Additional networks for use in application pods

DKP implements the ability to use additional software-defined networks (hereinafter referred to as additional networks) for application workloads (pods, virtual machines). You can use the following types of networks:

- Cluster (public) — a network that is publicly available in each project, configured and managed by the administrator. An example is a public WAN network or a shared network for traffic exchange between projects. To create such a network and use it for application pods, contact the cluster administrator.

- Project network (user network) — a network accessible within a namespace, created and managed by the user using the NetworkClass manifest provided by the administrator.

For more information about additional software-defined networks, see [Configuring and connecting additional virtual networks for use in application pods](#).

## Creating a project network (user network)

To create a network for a specific project, use the `Network` and `NetworkClass` custom resources provided to you by the administrator:

1. Create and apply the Network manifest by specifying the name of the NetworkClass obtained from the administrator in the `spec.networkClass` field:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: Network
metadata:
  name: my-network
  namespace: my-namespace
spec:
  networkClass: my-network-class # The name of the NetworkClass obtained from
  the administrator.
```

**i** Static identification of the VLAN ID number from the pool assigned by the cluster or network administrator is supported. If the value of the `spec.vlan.id` field is not specified, the VLAN ID will be assigned dynamically.

2. After creating the Network object you can check its status:

```
d8 k -n my-namespace get network my-network -o yaml
```

Example of the status of a Network object:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: Network
metadata:
  ...
status:
  bridgeName: d8-br-600
  conditions:
    - lastTransitionTime: "2025-09-29T14:51:26Z"
      message: All node interface attachments are ready
      reason: AllNodeInterfaceAttachmentsAreReady
      status: "True"
      type: AllNodeAttachmentsAreReady
    - lastTransitionTime: "2025-09-29T14:51:26Z"
      message: Network is operational
      reason: NetworkReady
      status: "True"
      type: Ready
  nodeAttachmentsCount: 1
  observedGeneration: 1
  readyNodeAttachmentsCount: 1
  vlanID: 600
```

After creating a network, you can connect it to pods.

## Connecting additional networks to pods

You can connect cluster networks and project networks to pods. To do this, use the pod annotation, specifying the parameters of the additional networks to be connected.

Example of a pod manifest with two additional networks added (the cluster network `my-cluster-`

network and the project network `my-network`):

**i** The `ifName` field (optional) specifies the name of the TAP interface within the subnet. The `mac` field (optional) specifies the MAC address to be assigned to the TAP interface.

```
apiVersion: v1
kind: Pod
metadata:
  name: app-with-additional-networks
  namespace: my-namespace
  annotations:
    network.deckhouse.io/networks-spec: |
      [
        {
          "type": "Network",
          "name": "my-network",
          "ifName": "veth_mynet",
          "mac": "aa:bb:cc:dd:ee:ff"
        },
        {
          "type": "ClusterNetwork",
          "name": "my-cluster-network",
          "ifName": "veth_public"
        }
      ]
spec:
  containers:
    - name: app
    # other parameters...
```

## IPAM for additional networks (IP allocation and assignment)

The IPAM (IP Address Management) 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.

The cluster administrator is responsible for allocating IP addresses for connection to cluster networks. They enable and configure IPAM for networks and define the IP address pool for them. Users can assign addresses and configure IPAM in project networks.

## Features of using IPAM in a DKP cluster

IPAM in a DKP cluster has the following usage features:

- IPAM is enabled **at the network level** via the `spec.ipam.ipAddressPoolRef` parameter (for ClusterNetwork IPAM, the cluster administrator enables it).
- IP address assignment to the pod interface is described in the annotation `network.deckhouse.io/networks-spec` added to the pod through the following fields:
- `ipAddressNames` — a list of IPAddress objects to be assigned to this interface (if the parameter is not specified, IPAddress can be created automatically).
  - `skipIPAssignment` — control of IPAddress reservation/tracking. If `skipIPAssignment: true`, IPAddress reservation/tracking is enabled, but the IP address is **not assigned** to the interface within the pod (advanced usage option).
- **Only IPv4** is supported.



If multiple additional networks with IPAM enabled are connected to a single pod, it is recommended to explicitly specify `ipAddressNames` for each interface (creating separate IPAddresses). Automatically created IPAddress is bound to the pod and may not be suitable for multiple IPAM networks at the same time.

## Allocating a pool of IP addresses for the project network and enabling IPAM

- i** To allocate a pool of addresses for the project network (Network), create an IPAddressPool resource **in the same namespace** as the project network (pods connected to the network).

To allocate a pool of addresses and assign them to network interfaces of pods connected to the project network, perform the following steps:

1. Create an address pool. To do this, use the IPAddressPool resource.

Example:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: IPAddressPool
metadata:
  name: my-net-pool
  namespace: my-namespace
spec:
  leaseTTL: 1h
  pools:
    - network: 192.168.10.0/24
      ranges:
        - 192.168.10.50-192.168.10.200
      routes:
        - destination: 10.10.0.0/16
          via: 192.168.10.1
```

- 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 IPAddressPool created in the previous step in the `spec.ipam.ipAddressPoolRef` parameter of the Network resource.

Example:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: Network
metadata:
  name: my-network
  namespace: my-namespace
spec:
  networkClass: my-network-class
  ipam:
    ipAddressPoolRef:
      kind: IPAddressPool
      name: my-net-pool
```

After allocating a pool of IP addresses for the project network, they can be assigned to the network interfaces of the pods connected to this network.

## Assigning IP addresses to network interfaces of pods connected to an additional network

DKP implements automatic IP address assignment for additional pod interfaces, as well as the ability to manually assign specific static IP addresses to additional pod interfaces.

### Automatic IP address assignment

To have the IP address for the additional network interface pod selected automatically from the pool, add the annotation `network.deckhouse.io/networks-spec` to the pod. In this annotation, specify the network parameters with IPAM enabled.

Example (the IP address will be automatically selected from the pool created for the `my-network`

network and assigned to the `net1` interface):

```
apiVersion: v1
kind: Pod
metadata:
  name: app-with-ipam
  namespace: my-namespace
  annotations:
    network.deckhouse.io/networks-spec: |
      [
        {
          "type": "Network",
          "name": "my-network",
          "ifName": "net1"
        }
      ]
spec:
  containers:
    - name: app
      image: nginx
```

In this case, an `IPAddress` object (type `Auto`) will be automatically created, and an IP address will be automatically selected from the pool attached to the additional network (in the example, `my-network`) and assigned to the pod network interface.

### Manual (explicit) creation of `IPAddress` with type `Auto`

You can also manually create an `IPAddress` object with `spec.type: Auto` (without specifying the `static.ip` parameter). In this case, the controller will allocate a free address from the pool attached to the additional network (in the example, `my-network`), and you can bind it to a specific interface using the `ipAddressNames` parameter in the `network.deckhouse.io/networks-spec` annotation.

Example:

1. Create an `IPAddress` object:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: IPAddress
metadata:
  name: app-net1-auto
  namespace: my-namespace
spec:
  networkRef:
    kind: Network      # or ClusterNetwork
    name: my-network
  type: Auto
```

2. Assign an IP address from the pool to the pod interface:

```

apiVersion: v1
kind: Pod
metadata:
  name: app-with-manual-auto-ip
  namespace: my-namespace
  annotations:
    network.deckhouse.io/networks-spec: |
      [
        {
          "type": "Network",
          "name": "my-network",
          "ifName": "net1",
          "ipAddressNames": ["app-net1-auto"]
        }
      ]
spec:
  containers:
    - name: app
      image: nginx

```

## Manually assigning a static IP address to an additional pod interface

To assign a specific static IP address to an additional pod interface, follow these steps:

1. Create an IPAddress in the pod namespace and specify which network it is intended for and which IP address is required:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: IPAddress
metadata:
  name: app-net1-static
  namespace: my-namespace
spec:
  networkRef:
    kind: Network
    name: my-network
  type: Static
  static:
    ip: 192.168.10.50

```

2. Connect the network to the pod and specify the created IPAddress in `ipAddressNames`:

```

apiVersion: v1
kind: Pod
metadata:
  name: app-with-static-ip
  namespace: my-namespace
  annotations:
    network.deckhouse.io/networks-spec: |
      [
        {
          "type": "Network",
          "name": "my-network",
          "ifName": "net1",
          "ipAddressNames": ["app-net1-static"]
        }
      ]
spec:
  containers:
    - name: app
      image: nginx

```

## Verification of IP address assignment to an Interface

To verify that an IP address is assigned to the interface, follow these steps:

1. Check the allocated address and phase at `IPAddress` (the phase should be `Allocated`):

```
d8 k -n my-namespace get ipaddress app-net1-static -O yaml
```

Output example:

| NAME             | PHASE     | AGE  | TYPE | KIND    | NAME  | ADDRESS      | NETWORK         |
|------------------|-----------|------|------|---------|-------|--------------|-----------------|
| ipaddress-auto-1 | Allocated | 4d1h | Auto | Network | mynet | 192.168.12.1 | 192.168.12.0/24 |
| ipaddress-auto-2 | Allocated | 4d1h | Auto | Network | mynet | 192.168.12.2 | 192.168.12.0/24 |

2. Check the pod annotation `network.deckhouse.io/networks-status` (including `ipAddressConfigs` and routes):

```
d8 k -n my-namespace get pod app-with-static-ip -O
jsonpath='{.metadata.annotations.network\.deckhouse\.io/networks-status}' |
jq
```

Output example:

```
[
  {
    "type": "Network",
    "name": "mynet",
    "ifName": "aabbcc",
    "mac": "ae:1c:68:7a:00:8f",
    "vlanID": 0,
    "ipAddressConfigs": [
      {
        "name": "ipaddress-auto-1",
        "address": "192.168.12.1",
        "network": "192.168.12.0/24"
      }
    ],
    "conditions": [
      {
        "type": "Configured",
        "status": "True",
        "lastTransitionTime": "2026-02-26T10:06:49Z",
        "reason": "InterfaceConfiguredSuccessfully",
        "message": ""
      },
      {
        "type": "Negotiated",
        "status": "True",
        "lastTransitionTime": "2026-02-26T10:06:49Z",
        "reason": "Up",
        "message": ""
      }
    ]
  }
]
```

## Additional resources

- Module `sdn` custom resources

## Connecting physical network interfaces to DPDK application pods

If your namespace (project) hosts high-performance workloads that require direct access to hardware (e.g., DPDK applications), you can use direct connection of physical network interfaces (Physical Functions and Virtual Functions) to pods via Kubernetes Dynamic Resource Allocation (DRA).

Physical network interfaces can be connected to pods in one of two modes:

- **Shared**: Virtual Functions (VF) are created from Physical Functions (PF) using SR-IOV, and multiple pods can share the same hardware.
- **Dedicated**: Each pod gets exclusive access to the entire PF.

For more information about the capabilities and features of working with Underlay networks in DKP, see the section [Configuring and connecting underlay networks for hardware device forwarding](#).

### Connecting physical network interfaces to pods

To use physical network interfaces (PF/VF) directly in pods for DPDK applications, you need to:

1. Ensure that your namespace has been marked by the administrator for use with UnderlayNetwork.
2. Create a pod with an annotation requesting the Underlay network device.

### Creating a pod with a device from the Underlay network

Create a pod that requests a device from an UnderlayNetwork. The pod annotation should specify:

- **type**: "UnderlayNetwork": Indicates this is a physical device request
- **name**: "underlay-network-name": The name of the UnderlayNetwork resource created by the administrator
- **bindingMode**: The binding mode for the device (VFIO-PCI, DPDK, or NetDev).

Example pod configuration for DPDK mode (universal mode that automatically selects the appropriate driver for the network adapter vendor):



```

apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: mydpdk
  annotations:
    network.deckhouse.io/networks-spec: |
      [
        {
          "type": "UnderlayNetwork",
          "name": "dpdk-shared-network",
          "bindingMode": "DPDK"
        }
      ]
spec:
  containers:
    - name: dpdk-container
      image: dpdk-app:latest
      securityContext:
        privileged: false
        capabilities:
          add:
            - NET_ADMIN
            - NET_RAW
            - IPC_LOCK
      volumeMounts:
        - mountPath: /hugepages
          name: hugepage
      resources:
        limits:
          hugepages-2Mi: 4Gi
          memory: 4Gi
          cpu: 4
        requests:
          cpu: 4
          memory: 4Gi
      command: ["/bin/sh", "-c", "sleep infinity"]
  volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages

```

**i**

For DPDK applications, it is important to:

- Configure `capabilities` (NET\_ADMIN, NET\_RAW, IPC\_LOCK) to run in non-privileged mode instead of using `privileged: true`
- Mount hugepages volumes, as DPDK requires hugepages for efficient memory management.



For VF devices in Shared mode, you can optionally specify a `vlanID` in the annotation to configure VLAN tagging on the VF:

```
network.deckhouse.io/networks-spec: |
[
  {
    "type": "UnderlayNetwork",
    "name": "dpdk-shared-network",
    "bindingMode": "VFIO-PCI",
    "vlanID": 100
  }
]
```

After creating the pod, verify that the device was allocated by checking the `network.deckhouse.io/networks-status` annotation:

```
d8 k -n mydpdk get pod dpdk-app -O
jsonpath='{.metadata.annotations.network\.deckhouse\.io/networks-status}' | jq
```

You can also check the ResourceClaim that was automatically created:

```
d8 k -n mydpdk get resourceclaim
```

Example pod status with allocated UnderlayNetwork device:

```

apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: mydpdk
  annotations:
    network.deckhouse.io/networks-spec: |
      [
        {
          "type": "UnderlayNetwork",
          "name": "dpdk-shared-network",
          "bindingMode": "DPDK"
        }
      ]
    network.deckhouse.io/networks-status: |
      [
        {
          "type": "UnderlayNetwork",
          "name": "dpdk-shared-network",
          "bindingMode": "DPDK",
          "netDevInterfaces": [
            {
              "name": "ens1f0",
              "mac": "00:1b:21:bb:aa:cc"
            }
          ],
          "conditions": [
            {
              "type": "Configured",
              "status": "True",
              "reason": "InterfaceConfiguredSuccessfully",
              "message": "",
              "lastTransitionTime": "2025-01-15T10:35:00Z"
            },
            {
              "type": "Negotiated",
              "status": "True",
              "reason": "Up",
              "message": "",
              "lastTransitionTime": "2025-01-15T10:35:00Z"
            }
          ]
        }
      ]
status:
  phase: Running
  ...

```

## Intra-cluster communication

To organize intra-cluster communication in Deckhouse Kubernetes Platform, it is recommended to use Services instead of accessing Pods directly. Services provide load balancing between Pods, stable network connectivity, and DNS integration for convenient service discovery. They also support various access scenarios and ensure isolation and security of network traffic.

If necessary, you can use either the standard Service-based load balancer or the advanced load balancer based on the `service-with-healthchecks` module.

### Standard load balancer

In Kubernetes, the Service resource is responsible for both internal and external request load balancing. This resource:

- Distributes requests between the application's working Pods.
- Excludes unhealthy Pods from load balancing.

Readiness probes specified in the specification of the containers belonging to a Pod are used to check whether the Pod is able to handle incoming requests.

## Limitations of the standard Service load balancer

The standard Service load balancing mechanism is suitable for most cloud application scenarios but has two limitations:

- If at least one container in a Pod fails the readiness probe, the entire Pod is considered `NotReady` and is excluded from load balancing for all Services it is associated with.
- Each container can only have one probe configured, so you cannot set up separate probes, for example, for read and write availability checks.

Example scenarios where the standard load balancer is insufficient:

- Database:
  - Runs in three Pods: `db-0`, `db-1`, and `db-2`, each containing a single container running the database process.
  - You need to create two Services: `db-write` for writing and `db-read` for reading.
  - Read requests must be balanced across all Pods.
  - Write requests must be balanced only to the Pod designated as the master by the database.
- Virtual machine:
  - A Pod contains a single container running the `qemu` process, acting as a hypervisor for a guest virtual machine.
  - Independent processes, such as a web server and an SMTP server, run on the guest VM.
  - You need to create two Services: `web` and `smtp`, each with its own readiness probe.

## Example Service for a standard load balancer

```
apiVersion: v1
kind: Service
metadata:
  name: productpage
  namespace: bookinfo
spec:
  ports:
    - name: http
      port: 9080
  selector:
    app: productpage
  type: ClusterIP
```

## Example of a standard load balancer for a PostgreSQL cluster

### Creating a StatefulSet for PostgreSQL

For a StatefulSet to work correctly, you need to create a standard Service to generate DNS names for individual Pods. This Service will not be used for direct database access.

```
apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  selector:
    app: postgres
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
```

Example 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

```

## Advanced load balancer

The advanced load balancer is implemented using the `ServiceWithHealthcheck` resources from the `service-with-healthchecks` module.

Unlike the standard load balancer where readiness probes are tied to container states, the `ServiceWithHealthcheck`-based load balancer allows you to configure active probes for individual TCP ports. This way, each load balancer serving the same Pod can operate independently from the others.

## Load balancer structure

The load balancer consists of two components:

- Controller — runs on cluster master nodes and manages `ServiceWithHealthcheck` resources.
- Agents — run on every cluster node and perform probes for Pods running on that node.

The `ServiceWithHealthcheck` load balancer is designed to be CNI-independent while using standard `Service` and `EndpointSlice` resources:

- When a `ServiceWithHealthcheck` resource is created, the controller automatically creates a `Service` with the same name in the same namespace, but with an empty `selector` field. This prevents the standard controller from creating `EndpointSlice` objects, which are normally used for load balancing.
- When an agent detects Pods on its node that fall under a `ServiceWithHealthcheck`, it runs the configured probes and creates an `EndpointSlice` with the list of verified IP addresses and ports. This `EndpointSlice` is linked to the `Service` created earlier.
- The CNI matches all `EndpointSlice` objects with the standard `Services` created earlier and performs load balancing across the verified IP addresses and ports on all cluster nodes.

Migrating from a `Service` to a `ServiceWithHealthchecks` resource, for example in a CI/CD pipeline, should not cause difficulties. The `ServiceWithHealthchecks` specification mostly repeats the standard `Service` specification, but includes an additional `healthcheck` section. During the resource lifecycle, a `Service` with the same name is created in the same namespace to route traffic to workloads in the cluster in the usual way (via `kube-proxy` or CNI).

## Configuring the load balancer



To use advanced load balancers:

- Configure the network policies for the project where ServiceWithHealthchecks load balancers will be used.
- If necessary, manually switch to using ServiceWithHealthchecks instead of standard load balancers (replace the existing Service objects with ServiceWithHealthchecks).

For more details, see the `service-with-healthchecks` module documentation.

You can configure this type of load balancing using the ServiceWithHealthchecks resource:

- Its specification is identical to a standard Service with the addition of a `healthcheck` section that contains a set of checks.
- Currently, three types of probes are supported:
  - `TCP`: A basic check using TCP connection establishment.
  - `HTTP`: Sends an HTTP request and expects a specific response code.
  - `PostgreSQL`: Sends an SQL query and expects it to complete successfully.

## Example configuration of advanced ServiceWithHealthchecks load balancers

Create a Secret to store the credentials required for database probe access:

```
d8 k -n my-ns create secret generic cred-secret --from-literal=user=postgres --from-literal=password=example cred-secret
```

Example 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
        postgresSQL:
          targetPort: 5432
          dbName: postgres
          authSecretName: cred-secret
          query: "SELECT 1"
```

Example 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
        postgresSQL:
          targetPort: 5432
          dbName: postgres
          authSecretName: cred-secret
          query: "SELECT NOT pg_is_in_recovery()"
```

## Hosting two independent load balancers on a single virtual machine

A Linux-based virtual machine runs two applications: an HTTP server (TCP 8080) and an SMTP server (TCP 2525). You need to configure two separate load balancers for these services — a web load balancer and an SMTP load balancer.

### Creating the virtual machine

Create a `my-vm` virtual machine based on the examples in the DVP documentation.

In the following manifest example, the label `vm: my-vm` is added for further identification in the load balancers.

```

apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachine
metadata:
  name: my-vm
  namespace: my-ns
  labels:
    vm: my-vm
spec:
  virtualMachineClassName: host
  cpu:
    cores: 1
  memory:
    size: 1Gi
  provisioning:
    type: UserData
    userData: |
      #cloud-config
      ssh_pwauth: True
      users:
        - name: cloud
          passwd:
'$6$rounds=4096$saltsalt$fPmUsbjAuA7mnQNTajQM6ClhesyG0.yyQhvahas02ejfMAq1ykBo1RquzS0
R6GgdIDlvS.kbUwDablGZKZcTP/'
      shell: /bin/bash
      sudo: ALL=(ALL) NOPASSWD:ALL
      lock_passwd: False
  blockDeviceRefs:
    - kind: VirtualDisk
      name: linux-disk

```

## Manifests for advanced load balancers for the web and SMTP services

Example web load balancer manifest:

```

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

```

Example SMTP load balancer manifest:



```

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

```

## Additional resources

- Module `service-with-healthchecks` documentation

## Managing authorization and access to workload with Istio

You can use the `istio` module to manage authorization and control access to workloads. Before configuring authorization, make sure the module is enabled in the cluster.

### Authorization

Authorization is managed using the `AuthorizationPolicy` resource from Istio. When this resource is created for a `Service`, the following request decision rules apply:

- If a request matches a `DENY` policy — deny the request.
- If there are no `ALLOW` policies for the `Service` — allow the request.
- If a request matches an `ALLOW` policy — allow the request.
- All other requests — to be denied.

In other words, if something is explicitly denied, the deny rule takes precedence. If something is explicitly allowed, only explicitly permitted requests are allowed (denies still have priority).

You can use the following arguments when writing authorization rules:

- Service identifiers and wildcards based on them ( `mycluster.local/ns/myns/sa/myapp` or `mycluster.local/*` )
- Namespace
- IP ranges
- HTTP headers
- JWT tokens from application requests

### AuthorizationPolicy resource

For more details on `AuthorizationPolicy`, refer to the `Istio` documentation.

The `AuthorizationPolicy` resource enables and defines access control to workloads. It supports both `ALLOW` and `DENY` rules described above.

Arguments for making authorization decisions:

- `source:`
  - `namespace`
  - `principal` (user identifier obtained after authentication)
  - `IP`
- `destination:`
  - `method` ( `GET`, `POST`, etc.)
  - `host`
  - `port`
  - `URI`
- `conditions:`

- HTTP headers
- `source` arguments
- `destination` arguments
- JWT tokens

## Additional resources

- Module `istio` documentation

## Configuring request routing with Istio

You can use the `istio` module in Deckhouse Kubernetes Platform to route HTTP and TCP requests.

The main resource for managing routing is `VirtualService` from Istio, which allows you to configure routing for HTTP or TCP requests.

## VirtualService resource

For more details on `VirtualService`, refer to the Istio documentation.

Using `VirtualService` is optional. Standard Services will continue to work if their functionality is sufficient. With this resource, you can configure request routing:

- Arguments for making routing decisions:
  - `host`
  - `uri`
  - `weight`
- Parameters of the final destinations:
  - New `host`
  - New `uri`
  - If the `host` is defined via a `DestinationRule`, requests can be routed to subsets
  - Timeout and retry settings



For `destination` in Istio to work correctly, it must be specified explicitly. If you are using an external API, specify it with a `ServiceEntry`.

## Additional resources

- Module `istio` documentation

## Enabling Istio for applications

Enabling Istio for applications is possible if the `istio` module is enabled and configured in the cluster. This is handled by the cluster administrator.

The essence of enabling Istio is adding a sidecar container to the application Pods, after which Istio can manage the traffic.

The recommended way to add sidecars is to use the sidecar injector. Istio can “inject” a sidecar container into application Pods using the Admission Webhook mechanism. Labels and annotations are used to add sidecars:

- A label on a `namespace` marks your namespace for the sidecar injector component. After the label is applied, new Pods will have sidecar containers added:
  - `istio-injection=enabled`: Uses the global Istio version ( `spec.settings.globalVersion` in the `ModuleConfig` resource);
  - `istio.io/rev=v1x16`: Uses a specific Istio version for this namespace.
- A Pod annotation `sidecar.istio.io/inject` ( `"true"` or `"false"` ) allows you to locally override the `sidecarInjectorPolicy`. These annotations only work in namespaces marked with one of the labels listed above.

It is also possible to add a sidecar to a specific Pod in a namespace without the `istio-injection=enabled` or `istio.io/rev=vXYZ` labels by setting the label `sidecar.istio.io/inject=true`.

The Istio proxy, which runs as a sidecar container, also consumes resources and adds overhead:

- Each request is DNATed into Envoy, which processes the request and creates another one. On the receiving side, the same happens.
- Each Envoy stores information about all Services in the cluster, which requires memory. The larger

the cluster, the more memory Envoy consumes. The solution is the Sidecar custom resource.

It is also important to prepare the Ingress controller and the application's Ingress resources:

- Enable `enableIstioSidecar` in the `IngressNginxController` resource.
- Add annotations to the application's Ingress resources:
  - `nginx.ingress.kubernetes.io/service-upstream: "true"`: The Ingress controller uses the Service's ClusterIP as the upstream instead of the Pod IPs. Traffic balancing between 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. Without this annotation, the controller will keep the site address header, for example, `Host: example.com`.

## Additional resources

- Module `ingress-nginx` custom resources
- Module `istio` documentation

## Managing request balancing between service endpoints with Istio

You can use the `istio` module to manage request load balancing between service endpoints. Before configuring load balancing, make sure the module is enabled in the cluster.

The main resource for managing request load balancing is `DestinationRule` from Istio. This resource allows you to configure the following parameters:

- TCP limits and timeouts.
- Algorithms for load balancing between endpoints.
- Rules for detecting endpoint issues and removing them from the load balancer.
- Encryption settings.



All configured limits apply separately to each client Pod. For example, if you set a limit of one TCP connection for a Service and there are three client Pods, the Service will receive three incoming connections.

## DestinationRule resource

For more details on `DestinationRule`, refer to the Istio documentation. Use this resource to:

- Define a traffic load balancing strategy between service endpoints:
  - Load balancing algorithm ( `LEAST_CONN`, `ROUND_ROBIN`, etc.).
  - Detection and exclusion of unhealthy endpoints.
  - TCP connection and request limits for endpoints.
  - Sticky Sessions support.
  - Circuit Breaker configuration.
- Define alternative endpoint groups for handling traffic (applicable to Canary Deployments). Each group can have its own load balancing strategy.
- Configure TLS for outgoing requests.

## Additional resources

- Module `istio` documentation

## Configuring Locality failover with Istio

In Deckhouse Kubernetes Platform, you can implement the Locality failover mechanism using the `istio` module. Before configuring the mechanism, make sure the module is enabled in the cluster.

The Locality failover mechanism manages traffic routing and directs it to a priority failover in case certain service instances become unavailable.



If necessary, refer to the Locality failover documentation.

With Istio, you can configure priority-based geographic failover between endpoints. Zones are determined based on node labels in the following hierarchy:

- `topology.istio.io/subzone`
- `topology.kubernetes.io/zone`
- `topology.kubernetes.io/region`

This is useful for inter-cluster failover when used together with multicluster configurations.



To enable Locality Failover, use the `DestinationRule` resource, which must also include the `outlierDetection` parameter.

Example:

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: helloworld
spec:
  host: helloworld
  trafficPolicy:
    loadBalancer:
      localityLbSetting:
        enabled: true # Enable LF.
    outlierDetection: # Required.
      consecutive5xxErrors: 1
      interval: 1s
      baseEjectionTime: 1m
```

## Additional resources

- Module `istio` documentation

## Configuring request retries with Istio

You can use the `istio` module to configure request retries. Before configuring retries, make sure the module is enabled in the cluster.

To configure request retries, use the `VirtualService` resource from Istio.



By default, when errors occur, all requests (including POST requests) are retried up to three times.

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
```

## VirtualService resource

If necessary, refer to the `VirtualService` documentation.

Using VirtualService is optional. Standard Services will continue to work if their functionality is sufficient. With this resource, you can configure request routing:

- Arguments for making routing decisions:
  - `host`
  - `uri`
  - `weight`
- Parameters of the final destinations:
  - New `host`
  - New `uri`
  - If the `host` is defined via a DestinationRule, requests can be routed to subsets
  - Timeout and retry settings



For `destination` in Istio to work correctly, it must be specified explicitly. If you are using an external API, specify it with a ServiceEntry.

## Additional resources

- Module `istio` documentation

## Configuring resources for istio-proxy sidecars

When using the `istio` module in a cluster, you can manage the resources allocated for istio-proxy sidecars in specific workloads. Annotations are used for this purpose.

### Supported annotations

To override global resource limits for istio-proxy sidecars, the following annotations are supported in individual workloads:

| Annotation                                     | Description                  | Example value      |
|------------------------------------------------|------------------------------|--------------------|
| <code>sidecar.istio.io/proxyCPU</code>         | CPU request for a sidecar    | <code>200m</code>  |
| <code>sidecar.istio.io/proxyCPULimit</code>    | CPU limit for a sidecar      | <code>"1"</code>   |
| <code>sidecar.istio.io/proxyMemory</code>      | Memory request for a sidecar | <code>128Mi</code> |
| <code>sidecar.istio.io/proxyMemoryLimit</code> | Memory limit for a sidecar   | <code>512Mi</code> |



All annotations from the table must be specified in the workload manifest at the same time. Partial configuration is not supported.

## 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 the manifest.
```

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 the manifest.

```

For Pods:

```

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 the manifest.

```

## Additional resources

- Module `istio` documentation

## Configuring Circuit Breaker

In Deckhouse Kubernetes Platform, the Circuit Breaker mechanism is implemented using Istio (the `istio` module) and provides the following capabilities:

- Temporarily exclude an endpoint from load balancing if the error limit is exceeded.
- Configure limits on the number of TCP connections and the number of requests to a single endpoint.
- Detect stuck requests and terminate them with an error code (HTTP request timeout).

## Example Circuit Breaker configuration

To detect problematic endpoints, use the `outlierDetection` settings in the `DestinationRule` custom resource. The Outlier Detection algorithm is described in more detail in the Envoy documentation.

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 # Maximum number of connections to the host, total across
all endpoints.
      http:
        maxRequestsPerConnection: 10 # The connection will be recreated after every 10
requests.
    outlierDetection:
      consecutive5xxErrors: 7 # Allows up to 7 errors (including `5xx`, TCP timeouts,
and HTTP timeouts)
      interval: 5m # within 5 minutes,
      baseEjectionTime: 15m # after which the endpoint will be removed from load
balancing for 15 minutes.

```

You can also use the `VirtualService` resource to configure HTTP timeouts. These timeouts are also taken into account when calculating endpoint error statistics.

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

```

## Additional resources

- Module `istio` documentation

## Configuring Canary deployment

Canary deployment is an application deployment strategy that allows gradually rolling out a new application version to production. This approach makes it possible to test new versions on a small portion of traffic, minimizing risks and ensuring a smooth transition. With Canary deployment, traffic can be shifted to the new version as confidence in its stability grows, with the ability to quickly roll back to the old version if issues arise. In Deckhouse Kubernetes Platform, Canary deployment can be implemented using the `ingress-nginx` module or the `istio` module (recommended).

### Example Canary deployment configuration with Ingress NGINX

To implement Canary deployment with Ingress NGINX, annotations and rules are used to route a portion of traffic to the new application version.

### Creating a Deployment and Service for the stable version

Example manifest for the stable version:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-v1
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
      version: v1
  template:
    metadata:
      labels:
        app: my-app
        version: v1
    spec:
      containers:
        - name: app
          image: app:v1
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: app-service
spec:
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 80
```

## Creating a Deployment and Service for the Canary version

Example manifest for the Canary version:



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
      version: v2
  template:
    metadata:
      labels:
        app: my-app
        version: v2
    spec:
      containers:
        - name: app
          image: app:v2
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: app-canary-service
spec:
  selector:
    app: my-app
    version: v2
  ports:
    - port: 80
      targetPort: 80
```

## Configuring Ingress for Canary deployment

For Canary deployment with Ingress NGINX, special annotations are used:

- `nginx.ingress.kubernetes.io/canary` : Enables Canary mode for the Ingress.
- `nginx.ingress.kubernetes.io/canary-weight` : Specifies the percentage of traffic to be routed to the Canary version.

Example Ingress manifest (10% of traffic will go to the Canary version ( `app-canary-service` ), 90% to the stable version ( `app-service` )):

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress
  annotations:
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "10" # 10% traffic to Canary.
spec:
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: app-canary-service
            port:
              number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress-main
spec:
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: app-service
            port:
              number: 80

```

## Gradually increasing traffic to the Canary version

You can gradually increase the percentage of traffic to the Canary version by changing the value of the `nginx.ingress.kubernetes.io/canary-weight` annotation. For example, to send 50% of traffic to the Canary version, update the annotation as follows:

```
nginx.ingress.kubernetes.io/canary-weight: "50"
```

## Rolling back or completing the Canary deployment

If the Canary version is stable, you can fully switch traffic to the new version by removing the Canary annotations and updating the main Ingress. If problems occur, you can reduce the percentage of traffic going to the Canary version or disable it completely by setting `nginx.ingress.kubernetes.io/canary-weight: "0"`.

## Additional annotations for Canary deployment

- `nginx.ingress.kubernetes.io/canary-by-header`: Routes traffic to the Canary version based on the value of an HTTP header.
- `nginx.ingress.kubernetes.io/canary-by-cookie`: Routes traffic to the Canary version based on the value of a cookie.

Example using a header:

```
nginx.ingress.kubernetes.io/canary-by-header: "canary"
nginx.ingress.kubernetes.io/canary-by-header-value: "true"
```

In this case, traffic will be routed to the Canary version if the request contains the header `canary: true`.

## Example Canary deployment configuration with Istio

**i** Istio is responsible only for flexible request routing based on special request headers (for example, cookies) or simple randomness. The actual routing configuration and switching between Canary versions is managed by the CI/CD system.

It is assumed that the same namespace contains two Deployments with different application versions. Pods of different versions have different labels (`version: v1` and `version: v2`).

Two custom resources need to be configured:

- DestinationRule describing how to identify different versions of your application (subsets).
- VirtualService describing how to distribute traffic between different application versions.

Example:

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: productpage-canary
spec:
  host: productpage
  # Subsets are available only when accessing the host through a VirtualService from
  # a Pod managed by Istio.
  # These subsets must be specified in the routes.
  subsets:
    - name: v1
      labels:
        version: v1
    - name: v2
      labels:
        version: v2
```

## Distribution based on the presence of a cookie

```
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 # Reference to a subset from DestinationRule.
    - route:
        - destination:
            host: productpage
            subset: v1
```

## Distribution based on probability

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: productpage-canary
spec:
  hosts:
  - productpage
  http:
  - route:
    - destination:
        host: productpage
        subset: v1 # Reference to a subset from DestinationRule.
        weight: 90 # Percentage of traffic that goes to Pods with the label `version: v1`.
    - route:
    - destination:
        host: productpage
        subset: v2
        weight: 10
```

## Additional resources

- Module `ingress-nginx` documentation
- Module `istio` documentation

## Configuration of container image vulnerability scanning

Deckhouse Kubernetes Platform (DKP) follows the CIS Kubernetes Benchmark recommendations, ensuring security both at the component level and across the platform as a whole.

To continuously monitor CIS compliance, DKP automatically runs checks in every cluster. The results of these checks are presented in reports and available in Grafana on the `Security / CIS Kubernetes Benchmark` dashboard.

DKP provides a built-in tool for automated container image vulnerability scanning based on Trivy.

Below are commands for viewing and filtering reports from vulnerability scans and CIS compliance checks performed in the cluster.

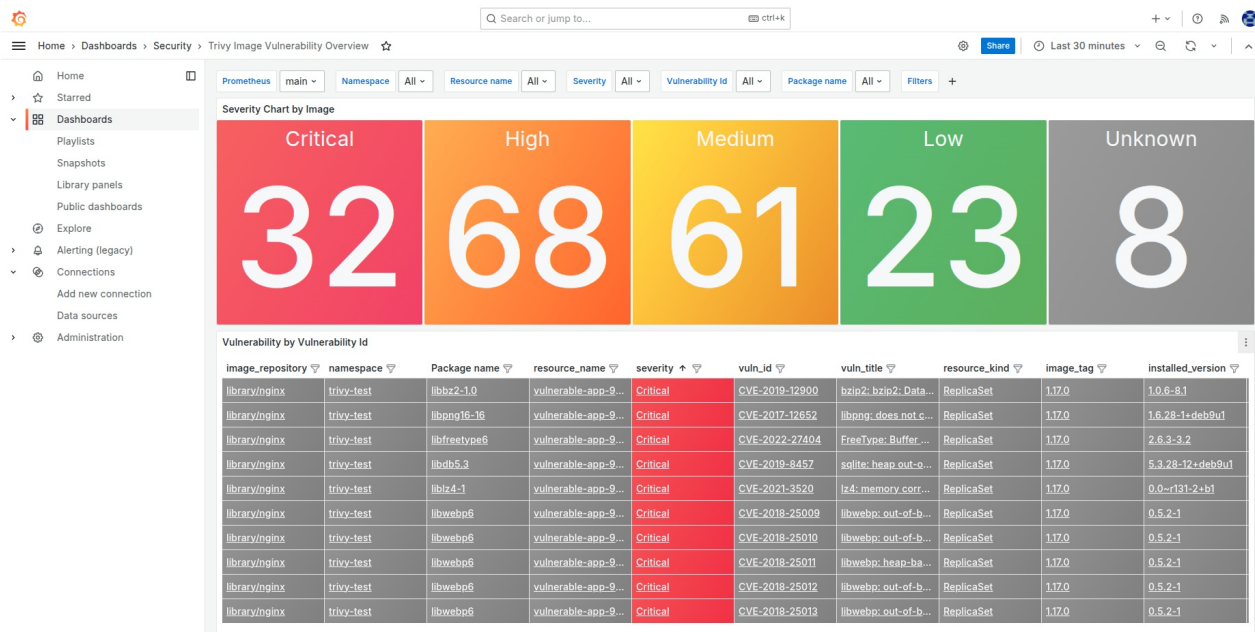
## Accessing scan results

Access to scan results, including the ability to view resources with reports, is granted to users with the following access roles:

- `d8:manage:networking:viewer` or higher
- `d8:manage:permission:module:operator-trivy:view`

## Viewing a scan report for your application

To view scan results for your application, use the Grafana dashboard `Security / Trivy Image Vulnerability Overview`. You can filter the results by namespace and resource.



## Viewing CIS compliance check results

- To list all resources that failed the check, run:

```
d8 k get clustercompliancereports.aquasecurity.github.io cis -ojson |
jq '.status.detailReport.results | map(select(.checks | map(.success) | all | not))'
```

- To search by a specific check ID:

```
check_id="5.7.3"
d8 k get clustercompliancereports.aquasecurity.github.io cis -ojson |
jq --arg check_id "$check_id" '.status.detailReport.results | map(select(.id == $check_id))'
```

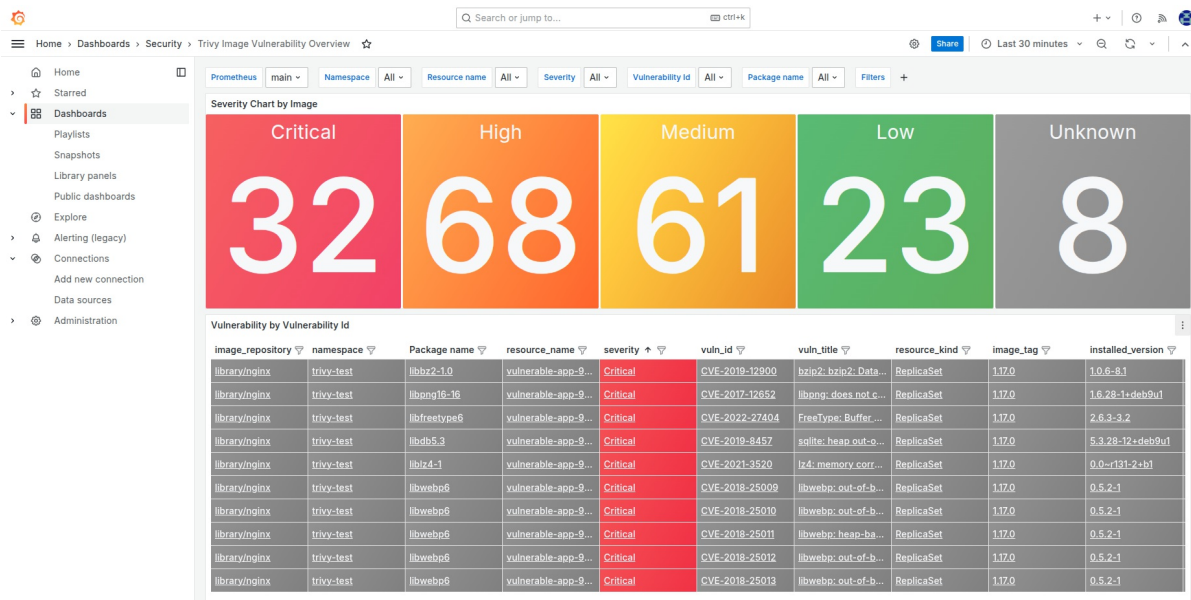
- To search by check description:

```
check_desc="Apply Security Context to Your Pods and Containers"
d8 k get clustercompliancereports.aquasecurity.github.io cis -ojson |
jq --arg check_desc "$check_desc" '.status.detailReport.results |
map(select(.description == $check_desc))'
```

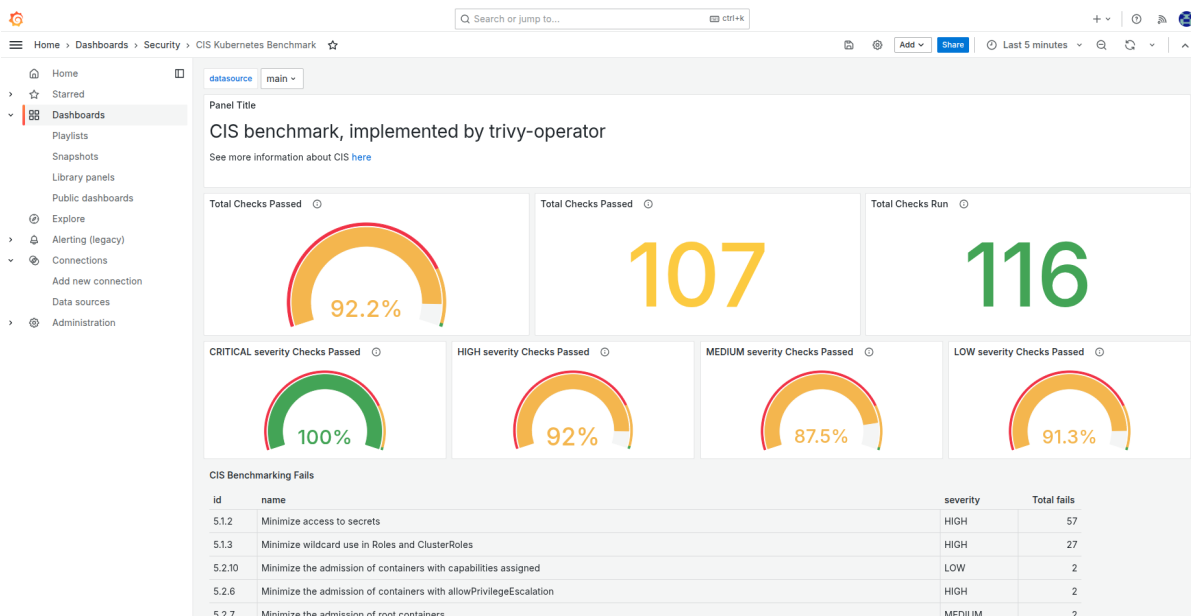
## Viewing scan results

Grafana dashboards:

- Security / Trivy Image Vulnerability Overview:** A summary of vulnerabilities in container images running in the cluster.



- **Security / CIS Kubernetes Benchmark:** Results of cluster compliance checks against the CIS Kubernetes Benchmark.



In the cluster:

- Cluster security reports:
  - ClusterComplianceReport
  - RbacAssessmentReport
- Workload security reports in the cluster:
  - VulnerabilityReport: Vulnerabilities in container images.
  - SbomReport: Software composition in images (SBOM).
  - ConfigAuditReport: Misconfigurations in Kubernetes objects.
  - ExposedSecretReport: Exposed secrets in containers.

DKP uses a set of custom resources developed by the Aqua Security Trivy Operator project to represent vulnerability scan results, configuration analysis, and cluster compliance checks.

Below is a description of the key CRDs created by operator-trivy, with examples and links to official documentation.

## Resource types for scan results

### Resource-level security

#### VulnerabilityReport

VulnerabilityReport is a resource containing a report of vulnerabilities detected in a container image used by a Kubernetes workload.

The report includes a list of known vulnerabilities in OS packages and applications, grouped by severity levels (Critical, High, Medium, etc.).

For each container in a multi-container workload, operator-trivy creates a separate `VulnerabilityReport` in the workload's namespace. The association with the Kubernetes resource is defined via the `ownerReference` field.

Resource names are formed based on the following pattern: `<workload-type>-<workload-name>-<container-name>`.

Example:

```

apiVersion: aquasecurity.github.io/v1alpha1
kind: VulnerabilityReport
metadata:
  name: replicaset-nginx-6d4cf56db6-nginx
  namespace: default
  labels:
    trivy-operator.container.name: nginx
    trivy-operator.resource.kind: ReplicaSet
    trivy-operator.resource.name: nginx-6d4cf56db6
    trivy-operator.resource.namespace: default
    resource-spec-hash: 7cb64cb677
  ownerReferences:
    - apiVersion: apps/v1
      kind: ReplicaSet
      name: nginx-6d4cf56db6
      uid: aa345200-cf24-443a-8f11-ddb438ff8659
      controller: true
      blockOwnerDeletion: false
report:
  artifact:
    repository: library/nginx
    tag: '1.16'
  os:
    family: debian
    name: '10.3'
  registry:
    server: index.docker.io
  scanner:
    name: Trivy
    vendor: Aqua Security
    version: 0.35.0
  summary:
    criticalCount: 2
    highCount: 0
    mediumCount: 0
    lowCount: 0
    unknownCount: 0
  vulnerabilities:
    - vulnerabilityID: CVE-2019-20367
      resource: libbsd0
      installedVersion: 0.9.1-2
      fixedVersion: 0.9.1-2+deb10u1
      severity: CRITICAL
      score: 9.1
      target: library/nginx:1.21.6
      primaryLink: https://avd.aquasec.com/nvd/cve-2019-20367
    - vulnerabilityID: CVE-2018-25009
      resource: libwebp6
      installedVersion: 0.6.1-2
      fixedVersion: ''
      severity: CRITICAL
      score: 9.1
      target: library/nginx:1.16
      title: 'libwebp: out-of-bounds read in WebPMuxCreateInternal'
      primaryLink: https://avd.aquasec.com/nvd/cve-2018-25009

```

## ConfigAuditReport

ConfigAuditReport is a resource containing results of Kubernetes object configuration checks performed with auditing tools such as Trivy.



The report includes a list of configuration findings, grouped by categories (for example, Security) and severity levels (Critical, High, etc.).

Examples of checks:

- Container runs as a non-root user.
- Container has resource limits.
- Network access configuration (hostNetwork, hostPID, etc.).
- Presence of security flags preventing privilege escalation.

A ConfigAuditReport is generated for any namespaced resource, including:

- Workloads (Pod, Deployment, StatefulSet, etc.)
- Auxiliary resources (Service, ConfigMap, Role, RoleBinding, etc.)

Each report is linked to the checked object via `ownerReference` and stored in the same namespace. Resource names are formed based on the following pattern: `<workload-type>-<workload-name>`.

Example:

```
apiVersion: aquasecurity.github.io/v1alpha1
kind: ConfigAuditReport
metadata:
  name: replicaset-nginx-6d4cf56db6
  namespace: default
  labels:
    trivy-operator.resource.kind: ReplicaSet
    trivy-operator.resource.name: nginx-6d4cf56db6
    trivy-operator.resource.namespace: default
ownerReferences:
  - apiVersion: apps/v1
    kind: ReplicaSet
    name: nginx-6d4cf56db6
report:
  scanner:
    name: Trivy
    vendor: Aqua Security
    version: '0.22.0'
  summary:
    criticalCount: 2
    highCount: 0
    mediumCount: 0
    lowCount: 9
  checks:
    - checkID: hostPIDSet
      severity: CRITICAL
      messages: ["Host PID is not configured"]
      success: true
    - checkID: notReadOnlyRootFilesystem
      severity: LOW
      messages: ["Filesystem should be read only"]
      success: false
      scope:
        type: Container
        value: nginx
```

## ExposedSecretReport

ExposedSecretReport is a resource containing results of scanning for potential secrets detected in a container image used by a Kubernetes workload.

The report lists strings containing sensitive data (for example, tokens, keys, passwords) found in files within the image. Each finding includes a category, rule, severity level, and file path.

For each container in a multi-container workload, operator-trivy creates a separate ExposedSecretReport in the workload's namespace. The association with the Kubernetes resource is

established via `ownerReference`.

Resource names are formed based on the following pattern: `<workload-type>-<workload-name>-<container-name>`.

Example:

```
apiVersion: aquasecurity.github.io/v1alpha1
kind: ExposedSecretReport
metadata:
  name: replicaset-app-67b77f5965-app
  namespace: default
  labels:
    trivy-operator.container.name: app
    trivy-operator.resource.kind: ReplicaSet
    trivy-operator.resource.name: app-67b77f5965
    trivy-operator.resource.namespace: default
  ownerReferences:
    - apiVersion: apps/v1
      kind: ReplicaSet
      name: app-67b77f5965
report:
  artifact:
    repository: myimagewithsecret
    tag: v0.22.0
  registry:
    server: index.docker.io
  scanner:
    name: Trivy
    vendor: Aqua Security
    version: 0.35.0
  secrets:
    - category: Stripe
      ruleID: stripe-access-token
      severity: HIGH
      target: "/app/config/secret.yaml"
      match: "publishable_key: *****"
      title: Stripe
    - category: Stripe
      ruleID: stripe-access-token
      severity: HIGH
      target: "/app/config/secret.yaml"
      match: "secret_key: *****"
      title: Stripe
  summary:
    criticalCount: 0
    highCount: 2
    mediumCount: 0
    lowCount: 0
  updateTimestamp: "2022-06-29T14:29:37Z"
```

## SbomReport

SbomReport is a resource containing a Software Bill of Materials (SBOM) report for a container image used by a Kubernetes workload.

It provides a complete list of software components, including system packages and application dependencies found in the container. This information is useful for image composition analysis, security auditing, and vendor compliance requirements.

For multi-container workloads, operator-trivy creates a separate SbomReport for each container. Reports are generated in the same namespace as the workload and linked via `ownerReference`.

Resource names are formed based on the following pattern: `<workload-type>-<workload-name>-`

<container-name>.

Example:

```
apiVersion: aquasecurity.github.io/v1alpha1
kind: SbomReport
metadata:
  creationTimestamp: "2023-07-10T09:37:21Z"
  generation: 1
  labels:
    resource-spec-hash: 796669cd5d
    trivy-operator.container.name: kube-apiserver
    trivy-operator.resource.kind: Pod
    trivy-operator.resource.name: kube-apiserver-kind-control-plane
    trivy-operator.resource.namespace: kube-system
  name: pod-kube-apiserver-kind-control-plane-kube-apiserver
  namespace: kube-system
  ownerReferences:
    - apiVersion: v1
      blockOwnerDeletion: false
      controller: true
      kind: Pod
      name: kube-apiserver-kind-control-plane
      uid: 732b4aa7-91f8-40a3-8b21-9627a98a910b
  resourceVersion: "6148"
  uid: 2a5000fe-b97e-46d0-9de7-62fb5fbc6555
report:
  artifact:
    repository: kube-apiserver
    tag: v1.21.1
  components:
    bomFormat: CycloneDX
    components:
      - bom-ref: 9464f5f9-750d-4ea0-8705-c8d067b25b29
        name: debian
        properties:
          - name: aquasecurity:trivy:Class
            value: os-pkgs
          - name: aquasecurity:trivy:Type
            value: debian
        supplier: {}
        type: operating-system
        version: "10.9"
      - bom-ref: pkg:deb/debian/base-files@10.3+deb10u9?arch=amd64&distro=debian-10.9
        licenses:
          - expression: GPL-3.0
            license: {}
        name: base-files
        properties:
          - name: aquasecurity:trivy:LayerDiffID
            value: sha256:417cb9b79adeec55f58b890dc9831e252e3523d8de5fd28b4ee2abb151b7dc8b
          - name: aquasecurity:trivy:LayerDigest
            value: sha256:5dea5ec2316d4a067b946b15c3c4f140b4f2ad607e73e9bc41b673ee5ebb99a3
          - name: aquasecurity:trivy:PkgeID
            value: base-files@10.3+deb10u9
          - name: aquasecurity:trivy:PkgeType
            value: debian
          - name: aquasecurity:trivy:SrcName
            value: base-files
          - name: aquasecurity:trivy:SrcVersion
            value: 10.3+deb10u9
        purl: pkg:deb/debian/base-files@10.3+deb10u9?arch=amd64&distro=debian-10.9
```

```

supplier:
  name: Santiago Vila <sanvila@debian.org>
  type: library
  version: 10.3+deb10u9
- bom-ref: pkg:deb/debian/netbase@5.6?arch=all&distro=debian-10.9
  licenses:
    - expression: GPL-2.0
      license: {}
  name: netbase
  properties:
    - name: aquasecurity:trivy:LayerDiffID
      value: sha256:417cb9b79adeec55f58b890dc9831e252e3523d8de5fd28b4ee2abb151b7dc8b
    - name: aquasecurity:trivy:LayerDigest
      value: sha256:5dea5ec2316d4a067b946b15c3c4f140b4f2ad607e73e9bc41b673ee5ebb99a3
    - name: aquasecurity:trivy:PkgID
      value: netbase@5.6
    - name: aquasecurity:trivy:PkgType
      value: debian
    - name: aquasecurity:trivy:SrcName
      value: netbase
    - name: aquasecurity:trivy:SrcVersion
      value: "5.6"
  purl: pkg:deb/debian/netbase@5.6?arch=all&distro=debian-10.9
  supplier:
    name: Marco d'Itri <md@linux.it>
    type: library
    version: "5.6"
- bom-ref: pkg:deb/debian/tzdata@2021a-0+deb10u1?arch=all&distro=debian-10.9
  name: tzdata
  properties:
    - name: aquasecurity:trivy:LayerDiffID
      value: sha256:417cb9b79adeec55f58b890dc9831e252e3523d8de5fd28b4ee2abb151b7dc8b
    - name: aquasecurity:trivy:LayerDigest
      value: sha256:5dea5ec2316d4a067b946b15c3c4f140b4f2ad607e73e9bc41b673ee5ebb99a3
    - name: aquasecurity:trivy:PkgID
      value: tzdata@2021a-0+deb10u1
    - name: aquasecurity:trivy:PkgType
      value: debian
    - name: aquasecurity:trivy:SrcName
      value: tzdata
    - name: aquasecurity:trivy:SrcRelease
      value: 0+deb10u1
    - name: aquasecurity:trivy:SrcVersion
      value: 2021a
  purl: pkg:deb/debian/tzdata@2021a-0+deb10u1?arch=all&distro=debian-10.9
  supplier:
    name: GNU Libc Maintainers <debian-glibc@lists.debian.org>
    type: library
    version: 2021a-0+deb10u1
  dependencies:
    - dependsOn:
        - pkg:deb/debian/base-files@10.3+deb10u9?arch=amd64&distro=debian-10.9
        - pkg:deb/debian/netbase@5.6?arch=all&distro=debian-10.9
        - pkg:deb/debian/tzdata@2021a-0+deb10u1?arch=all&distro=debian-10.9
      ref: 9464f5f9-750d-4ea0-8705-c8d067b25b29
    - dependsOn: []
      ref: pkg:deb/debian/base-files@10.3+deb10u9?arch=amd64&distro=debian-10.9
    - dependsOn: []
      ref: pkg:deb/debian/netbase@5.6?arch=all&distro=debian-10.9
    - dependsOn: []
      ref: pkg:deb/debian/tzdata@2021a-0+deb10u1?arch=all&distro=debian-10.9
    - dependsOn:

```

```

- 9464f5f9-750d-4ea0-8705-c8d067b25b29
  ref: pkg:oci/kube-
apiserver@sha256:53a13cd1588391888c5a8ac4cef13d3ee6d229cd904038936731af7131d193a9?
repository_url=k8s.gcr.io%2Fkube-apiserver&arch=amd64
  metadata:
    component:
      bom-ref: pkg:oci/kube-
apiserver@sha256:53a13cd1588391888c5a8ac4cef13d3ee6d229cd904038936731af7131d193a9?
repository_url=k8s.gcr.io%2Fkube-apiserver&arch=amd64
      name: k8s.gcr.io/kube-apiserver:v1.21.1
      properties:
        - name: aquasecurity:trivy:DiffID
          value:
sha256:417cb9b79adeec55f58b890dc9831e252e3523d8de5fd28b4ee2abb151b7dc8b, sha256:b5013
1762317bbe47def2d426d5c78a353a08b966d36bed4a04aee99dde4e12b, sha256:1e6ed7621dee7e03d
d779486ed469a65af6fb13071d13bd3a89c079683e3b1f0
        - name: aquasecurity:trivy:ImageID
          value:
sha256:771ffcf9ca634e37cbd3202fd86bd7e2df48ecba4067d1992541bfa00e88a9bb
        - name: aquasecurity:trivy:RepoDigest
          value: k8s.gcr.io/kube-
apiserver@sha256:53a13cd1588391888c5a8ac4cef13d3ee6d229cd904038936731af7131d193a9
        - name: aquasecurity:trivy:RepoTag
          value: k8s.gcr.io/kube-apiserver:v1.21.1
        - name: aquasecurity:trivy:SchemaVersion
          value: "2"
      purl: pkg:oci/kube-
apiserver@sha256:53a13cd1588391888c5a8ac4cef13d3ee6d229cd904038936731af7131d193a9?
repository_url=k8s.gcr.io%2Fkube-apiserver&arch=amd64
    supplier: {}
    type: container
    timestamp: "2023-07-10T09:37:21+00:00"
    tools:
      - name: trivy
        vendor: aquasecurity
    serialNumber: urn:uuid:50dbce86-28c5-4caf-9d08-a4aadf23233e
    specVersion: 1.4
    version: 1
  registry:
    server: k8s.gcr.io
  scanner:
    name: Trivy
    vendor: Aqua Security
    version: 0.52.2
  summary:
    componentsCount: 5
    dependenciesCount: 5
    updateTimestamp: "2023-07-10T09:37:21Z"

```

## Cluster-level security

### RbacAssessmentReport

RbacAssessmentReport is a resource containing a report generated from analyzing Kubernetes RBAC (Role-Based Access Control) settings.

It includes results of configuration audits performed with tools such as Trivy. Examples of findings include detection of roles that:

- Grant excessive privileges (for example, full access to secrets across all API groups).
- Violate the principle of least privilege.

Each report is associated with a specific Role or ClusterRole and stored in the same namespace as the

evaluated object.

Resource names follow the pattern: <Role|ClusterRole>-<role-name>.

Example:

```
apiVersion: aquasecurity.github.io/v1alpha1
kind: RbacAssessmentReport
metadata:
  name: role-868458b9d6
  namespace: kube-system
report:
  checks:
    - category: Kubernetes Security Check
      checkID: KSV051
      description: Check whether role permits creating role bindings and associating
        to privileged role/clusterrole
      messages:
        - ""
      severity: HIGH
      success: true
      title: Do not allow role binding creation and association with privileged
role/clusterrole
    - category: Kubernetes Security Check
      checkID: KSV056
      description: The ability to control which pods get service traffic directed to
        them allows for interception attacks. Controlling network policy allows for
        bypassing lateral movement restrictions.
      messages:
        - ""
      severity: HIGH
      success: true
      title: Do not allow management of networking resources
    - category: Kubernetes Security Check
      checkID: KSV041
      description: Check whether role permits managing secrets
      messages:
        - Role permits management of secret(s)
      severity: CRITICAL
      success: false
      title: Do not allow management of secrets
    - category: Kubernetes Security Check
      checkID: KSV047
      description: Check whether role permits privilege escalation from node proxy
      messages:
        - ""
      severity: HIGH
      success: true
      title: Do not allow privilege escalation from node proxy
    - category: Kubernetes Security Check
      checkID: KSV045
      description: Check whether role permits wildcard verb on specific resources
      messages:
        - ""
      severity: CRITICAL
      success: true
      title: No wildcard verb roles
    - category: Kubernetes Security Check
      checkID: KSV054
      description: Check whether role permits attaching to shell on pods
      messages:
        - ""
      severity: HIGH
```

```

    success: true
    title: Do not allow attaching to shell on pods
  - category: Kubernetes Security Check
    checkID: KSV044
    description: Check whether role permits wildcard verb on wildcard resource
    messages:
      - ""
    severity: CRITICAL
    success: true
    title: No wildcard verb and resource roles
  - category: Kubernetes Security Check
    checkID: KSV050
    description: An effective level of access equivalent to cluster-admin should
not    be provided.
    messages:
      - ""
    severity: CRITICAL
    success: true
    title: Do not allow management of RBAC resources
  - category: Kubernetes Security Check
    checkID: KSV046
    description: Check whether role permits specific verb on wildcard resources
    messages:
      - ""
    severity: CRITICAL
    success: true
    title: No wildcard resource roles
  - category: Kubernetes Security Check
    checkID: KSV055
    description: Check whether role permits allowing users in a rolebinding to add
      other users to their rolebindings
    messages:
      - ""
    severity: LOW
    success: true
    title: Do not allow users in a rolebinding to add other users to their
rolebindings
  - category: Kubernetes Security Check
    checkID: KSV052
    description: Check whether role permits creating role ClusterRoleBindings and
      association with privileged cluster role
    messages:
      - ""
    severity: HIGH
    success: true
    title: Do not allow role to create ClusterRoleBindings and association with
privileged
    role
  - category: Kubernetes Security Check
    checkID: KSV053
    description: Check whether role permits getting shell on pods
    messages:
      - ""
    severity: HIGH
    success: true
    title: Do not allow getting shell on pods
  - category: Kubernetes Security Check
    checkID: KSV042
    description: Used to cover attacker's tracks, but most clusters ship logs
quickly
    off-cluster.

```

```

    messages:
      - ""
    severity: MEDIUM
    success: true
    title: Do not allow deletion of pod logs
  - category: Kubernetes Security Check
    checkID: KSV049
    description: Some workloads leverage configmaps to store sensitive data or
configuration
      parameters that affect runtime behavior that can be modified by an attacker
      or combined with another issue to potentially lead to compromise.
    messages:
      - ""
    severity: MEDIUM
    success: true
    title: Do not allow management of configmaps
  - category: Kubernetes Security Check
    checkID: KSV043
    description: Check whether role permits impersonating privileged groups
    messages:
      - ""
    severity: CRITICAL
    success: true
    title: Do not allow impersonation of privileged groups
  - category: Kubernetes Security Check
    checkID: KSV048
    description: Check whether role permits update/create of a malicious pod
    messages:
      - ""
    severity: HIGH
    success: true
    title: Do not allow update/create of a malicious pod
  scanner:
    name: Trivy
    vendor: Aqua Security
    version: '0.22.0'
  summary:
    criticalCount: 1
    highCount: 0
    lowCount: 0
    mediumCount: 0
  updateTimestamp: null

```

## ClusterComplianceReport

ClusterComplianceReport is a resource containing a summary of cluster compliance with security requirements.

Currently, compliance checks include the CIS Kubernetes Benchmark, a set of best practices for securely configuring Kubernetes components.

Report structure:

- `spec.compliance.controls`: Description of the evaluated criteria.
- `status`: Results of the checks matching the description under `spec`. These results are based on aggregated reports from various security scanners.



You can also review the compliance report in the Grafana dashboard `Security / CIS Kubernetes Benchmark`.

Example:



```

apiVersion: aquasecurity.github.io/v1alpha1
kind: ClusterComplianceReport
metadata:
  name: cis
spec:
  compliance:
    controls:
      - checks:
          - id: AVD-KCV-0048
            commands:
              - id: CMD-0001
                description: Ensure that the API server pod specification file has permissions
                  of 600 or more restrictive
                id: 1.1.1
                name: Ensure that the API server pod specification file permissions are set
                  to 600 or more restrictive
                severity: HIGH
            - checks:
                - id: AVD-KCV-0049
                  commands:
                    - id: CMD-0002
                      description: Ensure that the API server pod specification file ownership is
                        set to root:root
                      id: 1.1.2
                      name: Ensure that the API server pod specification file ownership is set to
                        root:root
                      severity: HIGH
          ...
        summary:
          failCount: 9
          passCount: 107
          updateTimestamp: "2025-07-29T06:00:00Z"

```

## Additional resources

- Module `operator-trivy` documentation

## Configuration of using TLS certificates

Deckhouse Kubernetes Platform (DKP) provides built-in tools for managing TLS certificates, simplifying setup and management of traffic encryption for applications running in the cluster.

This page covers the following aspects of certificate usage in DKP:

- Manually requesting TLS certificates using the Certificate and ClusterIssuer resources.
- Securely storing and using credentials for accessing certificate authorities (CAs).
- Automatically obtaining certificates using the `tls-acme` annotation in Ingress resources.

**i**

For a general overview of certificate management in DKP, the list of supported issuers, and setup recommendations, refer to Certificate management.

## Working with certificates

### Viewing certificate details

- To list all certificates in the cluster, use the following command:

```
d8 k get certificate --all-namespaces
```

- To check the status of a specific certificate, run:

```
d8 k -n <NAMESPACE> describe certificate <CERTIFICATE-NAME>
```

## Automatic certificate requesting

To request a `letsencrypt` certificate, follow these steps:

1. Create a Certificate resource as described in the `cert-manager` documentation. Use the following example as a reference:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com          # Certificate name.
  namespace: default
spec:
  secretName: example-com-tls # Name of the Secret that will store the private
                             # key and the certificate.
  issuerRef:
    kind: ClusterIssuer      # Certificate issuer details.
    name: letsencrypt
  commonName: example.com    # Certificate main domain.
  dnsNames:                  # Optional additional certificate domains (at least
                             # one DNS name or IP address).
    - www.example.com
    - admin.example.com
```

2. The `cert-manager` module will automatically run a domain ownership challenge using the method defined in the ClusterIssuer resource (for example, `HTTP-01` or `DNS-01`).
3. The `cert-manager` module will automatically create a temporary Ingress resource for the challenge. This temporary Ingress will not affect your existing Ingress configuration.
4. Once validated, the issued certificate will be stored in the Secret specified in the `secretName` field.

**i** If you see a CAA record does not match issuer error during certificate issuance, check the DNS records for the domain. To use `letsencrypt`, your domain must have the following CAA record: `issue "letsencrypt.org"`.

For more details on CAA records, refer to Let's Encrypt documentation.

## Requesting a wildcard certificate using DNS in Cloudflare

1. Obtain your `GlobalAPIKey` and Email:
  - Go to `dash.cloudflare.com/profile`.
  - Your email is shown at the top under **Email Address**.
  - To view the key, click **View** next to **Global API Key** at the bottom.
2. Edit the `cert-manager` module settings, adding the following section:

```
settings:
  cloudflareGlobalAPIKey: APIkey
  cloudflareEmail: some@mail.somedomain
```

Or, use an API token instead (recommended):

```
settings:
  cloudflareAPIToken: some-token
  cloudflareEmail: some@mail.somedomain
```

DKP will automatically create a ClusterIssuer and Secret for Cloudflare in the `d8-cert-manager` namespace.

3. Create a Certificate resource using Cloudflare for DNS validation. This option becomes available only after the `cloudflareGlobalAPIKey` and `cloudflareEmail` parameters are configured:

```
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 the Ingress resource:

```
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
```

## Requesting a wildcard certificate using DNS in AWS Route53

1. Create a user with the required permissions:
  - On the policy management page, create a policy with the following permissions:

```
{
  "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": "*"
    }
  ]
}
```

- Open the user management page and create a user with the above policy.

2. Edit the `cert-manager` module settings to add the following section:

```
settings:
  route53AccessKeyID: <ACCESS_KEY_ID>
  route53SecretAccessKey: <SECRET_ACCESS_KEY>
```

DKP will automatically create a ClusterIssuer and Secret for Route53 in the `d8-cert-manager` namespace.

3. Create a Certificate resource for validation with Route53. This option becomes available only after the `route53AccessKeyID` and `route53SecretAccessKey` parameters are configured:

```
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"
```

## Requesting a wildcard certificate using DNS in Google

1. Create a ServiceAccount with the required role:
  - Open the policy management page.
  - Choose the target project and create a ServiceAccount under any name (for example, `dns01-solver`).
  - In the created account, create a key by clicking **Add key**. A JSON file with the key data will be downloaded.
  - Encode the JSON file as a Base64 string:

```
base64 project-209317-556c656b81c4.json
```

2. Save the Base64 string in the `cloudDNSServiceAccount` parameter.

DKP will automatically create a ClusterIssuer and Secret for CloudDNS in the `d8-cert-manager` namespace.

3. Create a Certificate resource using CloudDNS for validation:

```
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"
```

## Requesting a self-signed certificate

To issue a self-signed certificate, specify `selfsigned` as the value for `issuerRef.name`:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com          # Certificate name.
  namespace: default
spec:
  secretName: example-com-tls # Name of the Secret that will store the private key
                             # and the certificate.
  issuerRef:
    kind: ClusterIssuer      # Certificate issuer details.
    name: selfsigned
  commonName: example.com    # Certificate main domain.
  dnsNames:                  # Optional additional certificate domains (at least one
                             # DNS name or IP address).
    - www.example.com
    - admin.example.com
```

## Generating a self-signed certificate

When generating certificates manually, it is important to fill out all fields of the certificate request correctly to ensure that the final certificate is issued properly and can be validated across various services.

It is important to follow these guidelines:

1. Specify domain names in the `SAN` (Subject Alternative Name) field.

The `SAN` field is a more modern and commonly used method for specifying the domain names covered by the certificate. Some services no longer consider the `CN` (Common Name) field as the source for domain names.

2. Correctly fill out the `keyUsage`, `basicConstraints`, `extendedKeyUsage` fields, specifically:

- `basicConstraints = CA:FALSE`

This field determines whether the certificate is an end-entity certificate or a certification authority (CA) certificate. CA certificates cannot be used as service certificates.

- `keyUsage = digitalSignature, keyEncipherment`

The `keyUsage` field limits the permissible usage scenarios of this key:

- `digitalSignature`: Allows the key to be used for signing digital messages and

ensuring data integrity.

- `keyEncipherment`: Allows the key to be used for encrypting other keys, which is necessary for secure data exchange using TLS (Transport Layer Security).
- `extendedKeyUsage = serverAuth`

The `extendedKeyUsage` field specifies additional key usage scenarios required by specific protocols or applications:

- `serverAuth`: Indicates that the certificate is intended for server use, authenticating the server to the client during the establishment of a secure connection.

It is also recommended to:

1. Issue the certificate for no more than 1 year (365 days).

The validity period of the certificate affects its security. A one-year validity ensures the cryptographic methods remain current and allows for timely certificate updates in case of threats. Furthermore, some modern browsers now reject certificates with a validity period longer than 1 year.

2. Use robust cryptographic algorithms, such as elliptic curve algorithms (including `prime256v1`).

Elliptic curve algorithms (ECC) provide a high level of security with a smaller key size compared to traditional methods like RSA. This makes the certificates more efficient in terms of performance and secure in the long term.

3. Do not specify domains in the `CN` (Common Name) field.

Historically, the `CN` field was used to specify the primary domain name for which the certificate was issued. However, modern standards, such as RFC 2818, recommend using the `SAN` (Subject Alternative Name) field for this purpose. If the certificate is intended for multiple domain names listed in the `SAN` field, specifying one of the domains additionally in `CN` can cause a validation error in some services when accessing domains not listed in `CN`. If non-domain-related information is specified in `CN` (for example, an identifier or service name), the certificate will also extend to these names, which could be exploited for malicious purposes.

## Certificate generation example

To generate a certificate, we'll use the `openssl` utility.

1. Fill in the `cert.cnf` configuration file:

```
[ req ]
default_bits      = 2048
default_md        = sha256
prompt           = no
distinguished_name = dn
req_extensions    = req_ext
[ dn ]
C = GB
ST = London
L = London
O = Example Company
OU = IT Department
# CN = Do not specify the CN field.
[ req_ext ]
subjectAltName = @alt_names
[ alt_names ]
# Specify all domain names.
DNS.1 = example.co.uk
DNS.2 = www.example.co.uk
DNS.3 = api.example.co.uk
# Specify IP addresses (if required).
IP.1 = 192.0.2.1
IP.2 = 192.0.4.1
[ v3_ca ]
basicConstraints = CA:FALSE
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
# Elliptic curve parameters.
[ ec_params ]
name = prime256v1
```

2. Generate an elliptic curve key:

```
openssl ecparam -genkey -name prime256v1 -noout -out ec_private_key.pem
```

3. Create a certificate signing request:

```
openssl req -new -key ec_private_key.pem -out example.csr -config cert.cnf
```

4. Generate a self-signed certificate:

```
openssl x509 -req -in example.csr -signkey ec_private_key.pem -out example.crt -days 365 -
extensions v3_req -extfile cert.cnf
```

## Protecting credentials

If you prefer not to store credentials in the DKP configuration, you can create a separate Secret and reference it in the ClusterIssuer resource.

To do this, follow these steps:

1. Create a Secret with the access key:

```
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
```

2. Create a ClusterIssuer that references the 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
```

3. Request certificates as usual, using the 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
```

## Support for tls-acme annotation

DKP supports the annotation `kubernetes.io/tls-acme: "true"` in Ingress resources. The `cert-manager-ingress-shim` component watches for this annotation and automatically creates Certificate resources in the same namespace as the Ingress.





When using the annotation, the Certificate resource is linked to the existing Ingress. No separate Ingress is created for domain validation; instead, additional records are added to the existing Ingress. If the Ingress has authentication or a whitelist configured, the validation will fail. Therefore, it's recommended that you use a Certificate resource directly instead.

If switching from annotation to a manual Certificate, make sure to delete the automatically generated Certificate first. Otherwise, both will try to update the same Secret, potentially exceeding Let's Encrypt rate limits.

Example Ingress configuration with the annotation:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/tls-acme: "true"          # 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                  # 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                  # Additional domain.
        - admin.example.com                # Another additional domain.
      secretName: example-com-tls          # Name of the Certificate and Secret.
```

## Additional resources

- Module `cert-manager` parameters

# Deckhouse Kubernetes Platform web UI

The Deckhouse Kubernetes Platform (DKP) web UI provides access to cluster management, including monitoring, parameter configuration, node and module management, as well as security and network configuration. Most operations available from the command line through the `d8` utility (Deckhouse CLI) or `kubectx` can also be performed through the web UI.

## Accessing the web UI

To open the web UI, do as follows:

1. Enter `console.<CLUSTER_NAME_TEMPLATE>` in the browser address bar, where `<CLUSTER_NAME_TEMPLATE>` is a string corresponding to the cluster DNS name template specified in the global parameter `modules.publicDomainTemplate`. The address format may differ depending on the system configuration. Contact your administrator to find out the URL.
2. During the first login, enter the user credentials provided by the security administrator. After successful authentication, the main page of the web UI opens. The page contains the following sections:
  - “Deckhouse”
  - “Node management”
  - “Multitenancy”
  - “Access”
  - “Network”
  - “Storage”
  - “Security”
  - “Monitoring”
  - “Logging”

## Deckhouse

### “Overview” subsection

The “Overview” subsection contains key information about the Deckhouse Kubernetes Platform (DKP) cluster and its components.

The screenshot displays the Deckhouse web UI interface. On the left is a sidebar with a search bar and a menu containing sections like 'System Management', 'Pinned sections', 'Overview', 'Modules', 'Deckhouse', 'Releases', 'Global settings', 'Platform Operator', 'Kubernetes', 'Projects', 'Project Templates', 'Namespaces', 'Nodes', 'Node Groups', 'Node Group Configurations', 'Instance Classes', and 'Static Instances'. The main content area has a breadcrumb 'System / System Management / Deckhouse'. At the top are two large panels: 'Deckhouse EE v1.75.7 EarlyAccess' and 'Kubernetes v1.33.10'. Below these is a 'Tools' section with eight tiles: COMMANDER, KUBERNETES DASHBOARD, DOCUMENTATION, MONITORING, PROMETHEUS, STATUS PAGE, COMPONENT AVAILABILITY, and KUBECONFIG GENERATOR. The 'Warnings' section shows four status boxes: 'Nodes with errors' (0), 'Nodes waiting for update' (0), 'Vulnerable containers' (11), and 'Active alerts' (10). The 'Subsystem statuses' section lists four items: 'Deckhouse Availability' (No Data, No connection to the server), 'Control Plane' (No Data, The availability of Kubernetes control-plane), 'Deckhouse' (No Data, The availability of deckhouse and working hook), and 'Extensions' (No Data, The availability of extensions apps).

Main elements of the web UI:

- **“Deckhouse” and “Kubernetes” panels:** Display the current Kubernetes version and general platform information.
- **“Tools” panel:** Contains buttons for quick access to the following sections:
  - “Kubernetes dashboard”

- “Documentation”
- “Grafana” (metrics monitoring)
- “Prometheus” (metrics collection and storage)
- “Status page”
- “Component availability”
- “Kubeconfig generator”
- **“Warnings” panel:** Displays errors, pending updates, and active alerts.
- **“Node groups with issues” panel:** Analyzes node health and displays problematic groups.
- **“Subsystem statuses” panel:** Displays the state of cluster services.
- **Resource monitoring:** Includes charts and indicators showing load changes.
- **Sidebar menu:** Navigation through the main sections (the visibility of sections depends on the user access level).
- **User menu (bottom left):**
  - User information
  - Settings (changing system parameters and viewing the web UI version)
  - YAML editor for configuration management

## “Updates” subsection

The “Updates” subsection contains information about DKP releases.

System / System Management / Deckhouse

### Updates

[Releases](#) [Settings](#) [YAML](#)

**General Information** **Update windows**

Update mode: Auto-patch ⓘ

Disruptive changes approval: Default

| Version  | Status     | Released            | Installed           | Message                     |
|----------|------------|---------------------|---------------------|-----------------------------|
| v1.75.7  | DEPLOYED   | 15.05.2026 17:43:28 | 15.05.2026 18:43:39 | — <a href="#">Changelog</a> |
| v1.75.6  | SUPERSEDED | 08.05.2026 15:18:52 | 15.05.2026 18:43:39 | — <a href="#">Changelog</a> |
| v1.75.5  | SUPERSEDED | 29.04.2026 10:28:01 | 08.05.2026 16:49:06 | — <a href="#">Changelog</a> |
| v1.75.4  | SUPERSEDED | 21.04.2026 16:48:07 | 29.04.2026 12:58:09 | — <a href="#">Changelog</a> |
| v1.75.3  | SUPERSEDED | 09.04.2026 12:13:25 | 21.04.2026 19:18:13 | — <a href="#">Changelog</a> |
| v1.74.15 | SUPERSEDED | 30.03.2026 14:02:56 | 09.04.2026 12:13:27 | — <a href="#">Changelog</a> |
| v1.74.14 | SUPERSEDED | 23.03.2026 14:19:32 | 30.03.2026 14:02:56 | — <a href="#">Changelog</a> |
| v1.74.13 | SUPERSEDED | 13.03.2026 11:38:15 | 23.03.2026 15:49:35 | — <a href="#">Changelog</a> |
| v1.74.11 | SUPERSEDED | 06.03.2026 13:44:18 | 13.03.2026 11:38:15 | — <a href="#">Changelog</a> |
| v1.74.10 | SUPERSEDED | 02.03.2026 17:23:38 | 06.03.2026 13:44:19 | — <a href="#">Changelog</a> |
| v1.74.8  | SUPERSEDED | 20.02.2026 12:43:21 | 02.03.2026 17:53:39 | — <a href="#">Changelog</a> |

## “Modules” subsection

The “Modules” subsection lists enabled and disabled modules. A filter is available to search for a required module.

88

Deckhouse

Search

System Management

Pinned sections

Overview

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

V

?

+ YAML

System / System Management / Deckhouse

Modules

ModulesUpdate settings

Search

| Name                         | Status   | Version | Channel     | Update mode |
|------------------------------|----------|---------|-------------|-------------|
| adaptive-resource-managem... | DISABLED | v0.0.6  | EarlyAccess | AutoPatch   |
| admission-policy-engine      | ENABLED  | v1.75.7 | EarlyAccess | AutoPatch   |
| ai-models                    | DISABLED | —       | EarlyAccess | AutoPatch   |
| alb                          | DISABLED | —       | EarlyAccess | AutoPatch   |
| backup                       | DISABLED | —       | EarlyAccess | AutoPatch   |
| cert-manager                 | ENABLED  | v1.75.7 | EarlyAccess | AutoPatch   |
| chrony                       | ENABLED  | v1.75.7 | EarlyAccess | AutoPatch   |
| cilium-hubble                | DISABLED | v1.75.7 | EarlyAccess | AutoPatch   |
| cloud-provider-aws           | DISABLED | v1.75.7 | EarlyAccess | AutoPatch   |
| cloud-provider-azure         | DISABLED | v1.75.7 | EarlyAccess | AutoPatch   |
| cloud-provider-dvp           | ENABLED  | v1.75.7 | EarlyAccess | AutoPatch   |
| cloud-provider-dynamix       | DISABLED | v1.75.7 | EarlyAccess | AutoPatch   |
| cloud-provider-gcp           | DISABLED | v1.75.7 | EarlyAccess | AutoPatch   |
| cloud-provider-huaweicloud   | DISABLED | v1.75.7 | EarlyAccess | AutoPatch   |
| cloud-provider-openstack     | DISABLED | v1.75.7 | EarlyAccess | AutoPatch   |
| cloud-provider-vcd           | DISABLED | v1.75.7 | EarlyAccess | AutoPatch   |

“Global settings” subsection

This subsection provides access to critical DKP cluster settings:

- **“Global cluster settings”**: Configuration of the DNS name template and the list of tolerations.
- **“Global module settings”**: Configuration of the high availability mode, and fields for specifying IngressClass and StorageClass.
- **“HTTPS operating mode”**: Configuration of security certificates.
- **“Kubernetes control plane component resources”**: Configuration of the number of allocated CPU cores and memory size.

⚠

These settings affect cluster stability, security, and fault tolerance, so modify them with caution.

88

Deckhouse

Search

System Management

Pinned sections

Overview

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

Instance Classes

Static Instances

Config

Deschedulers

V

?

+ YAML

System / System Management / Deckhouse

Global settings

Global cluster settings

DNS name template

%s.a.ui.foxtrot-dev.ru

Use %s as the dynamic part of the string.  
Example: %s.kube.company.my

Custom toleration keys

Must be specified to allow the scheduler  
to place critical Deckhouse components  
on dedicated nodes, such as CNI and CSI components

Global module settings

High availability mode

AutoYesNo

High availability mode is automatically enabled for clusters with more than one master node. In other cases,  
the value is automatically determined as false

StorageClass for all Deckhouse  
components

replicated

If the value is not specified, the automatically  
detected global.discovery.defaultStorageClass is  
used. If it is not defined, then emptyDir is used.

The specified StorageClass is applied when the  
module is enabled. This parameter should only be  
used in exceptional situations.

Ingress class (Ingress class) used for  
Deckhouse modules

nginx

HTTPS Mode

Kubernetes control plane component resources

The tabs in the “Kubernetes” subsection contain basic information about the Kubernetes cluster.

The top screenshot shows the 'Kubernetes' configuration page. The left sidebar contains a search bar and a 'System Management' button. The main content area has tabs for 'Configuration', 'Cloud', and 'YAML'. The 'Configuration' tab is active, showing settings for 'Cluster Domain' (dev-cluster.local), 'Kubernetes Version' (Default 1.33), and 'Container runtime type' (ContainerdV2). Below these are 'Network Settings' for 'Cluster Service Subnet' (10.223.0.0/16), 'Cluster Pod Subnet' (10.112.0.0/16), and 'Pod subnet prefix on nodes' (24). There is a 'Proxy Settings' toggle and a 'Container Registry' section with fields for 'Address of the container registry with Deckhouse images' (registry.deckhouse.ru/deckhouse/ee) and 'Scheme' (https). An 'Authorization' section includes fields for 'Username' (license-token), 'Auth', 'Password', and 'Email'.

The bottom screenshot shows the 'Cloud' tab of the 'Kubernetes' configuration page. It features a 'DKP Cluster' toggle, a 'Layout' dropdown (Standard), and three toggle switches for 'Master Node Group', 'Instance Class', and 'Virtual Machine'. Below these are fields for 'Virtual Machine Class' (amd-epyc-gen-3), 'CPU' (6), 'Core Fraction' (20%), 'Memory' (12Gi), and 'Memory Size'.

## Node management

### “Node groups” subsection

The settings in this subsection let you manage Kubernetes node groups, monitor their state, and add new groups.

The node group creation form contains the following fields:

- “Name”
- “Node count”
- “Static instance selector” (cannot be changed after creation)
- “Label expressions”

The lower part of the form contains additional settings, including node update parameters, the node template, system parameters, and Chaos Monkey parameters, which can also be expanded for detailed configuration.

88

Deckhouse

Search

System Management

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

Instance Classes

V

?

+ YAML

System / System Management / Nodes

Create

Node Groups

Node Types

Name by substring

Node Type

K8s Version

State

By name

Use it as a template

Delete

billing-prom

Node Type

Group Priority

Kubernetes Version

Instance Class

CloudEphemeral

-

1.33

billing-prom

Node State

Autoscaling Parameters

Zones

Total Nodes

Ready

Current

Nodes per Zone

Required

Ordered

Reserve

Taints: 1

Labels: 1

CPU

8%

0.28/4 cores

RAM

15%

1.67/11.67 GB

Disk

12%

5.47/47.39 GB

88

Deckhouse

Search

System Management

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

Instance Classes

V

?

+ YAML

System / System Management / Nodes / Node Groups

Delete

billing-prom

Configuration

Nodes

Events

Monitoring

YAML

Group Status

Node Type

Group Priority

Kubernetes Version

Instance Class

CloudEphemeral

-

1.33

billing-prom

Node State

Autoscaling Parameters

Zones

Total Nodes

Ready

Current

Nodes per Zone

Required

Ordered

Reserve

Minimum nodes per zone\*

spec.cloudInstances.minPerZone

1

Instance class\*

spec.cloudInstances.classReference.kind

billing-prom

Maximum nodes per zone\*

spec.cloudInstances.maxPerZone

1

Autoscaler parameters

Node Updates

Deckhouse

Search

System Management

Overview

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

Instance Classes

Static Instances

Config

Deschedulers

Priority Classes

YAML

System / System Management / Nodes / Node Groups

Minimum nodes per zone\*  
spec.cloudInstances.minPerZone

1

Instance class\*  
spec.cloudInstances.classReference.kind

billing-prom

Maximum nodes per zone\*  
spec.cloudInstances.maxPerZone

1

Number of replicas in MachineDeployment and lower limit in cluster autoscaler

Autoscaler parameters

Node Updates

Node Template

Labels

KEY

VALUE

node-role/billing-pr...

+ ADD

Annotations

KEY

VALUE

+ ADD

Taints

Labels and expressions

dedicated billing-prom NoExecute

Add

Node System Settings

Chaos Monkey parameters

Deckhouse

Search

System Management

Overview

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

Instance Classes

Static Instances

Config

Deschedulers

Priority Classes

YAML

System / System Management / Nodes / Node Groups

Node System Settings

CRI Settings

ContainerdV2

Containerd

Not managed

By default

Topology Manager

The Topology Manager allows the Kubernetes scheduler to consider resource topology data when scheduling pods.  
In a cluster with NUMA nodes, the Topology Manager can improve application performance by placing pod containers on the NUMA node that works more efficiently with resources.  
More information in the Kubernetes documentation.

Kubelet Parameters

Maximum Number of Log Files with Rotation Considered  
spec.kubelet.containerLogMaxFiles

4

Maximum log file size before rotation  
spec.kubelet.containerLogMaxSize

50Mi

Maximum number of pods on a node  
spec.kubelet.maxPods

110

Path to Directory for Kubelet Files  
spec.kubelet.rootDir

/var/lib/kubelet

GPU Resource Configuration

The node group card displays:

- Node type and Kubernetes version
- Total number of nodes, their readiness and update status
- Load metrics (CPU, memory, disk)
- Taints and labels

billing-prom

Use it as a template

Delete

Node Type

Group Priority

Kubernetes Version

Instance Class

CloudEphemeral

—

1.33

[billing-prom](#)

Node State

Autoscaling Parameters

Zones

Total Nodes

1

Ready

1

Current

1

Nodes per Zone

1

Required

1

Ordered

1

Reserve

—

Taints: 1

Labels: 1

CPU

8%

0.28/4 cores

RAM

15%

1.66/11.67 GB

Disk

12%

5.47/47.39 GB

The “Create” button in the upper-right corner opens the form for creating a new node group.

Deckhouse

Search

System Management

Pinned sections

Overview

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

System / System Management / Nodes

Node Groups

Node Types

Name by substring

Node Type

K8s Version

State

By name

billing-prom

Use it as a template

Delete

Node Type

Group Priority

Kubernetes Version

Instance Class

CloudEphemeral

—

1.33

[billing-prom](#)

Node State

Autoscaling Parameters

Zones

Total Nodes

1

Ready

1

Current

1

Nodes per Zone

1

Required

1

Ordered

1

Reserve

—

Taints: 1

Labels: 1

CPU

7%

0.25/4 cores

RAM

15%

1.68/11.67 GB

Disk

12%

5.47/47.39 GB

master

Use it as a template

Delete

Node Type

Kubernetes Version

CloudPermanent

1.33

Node State

The form lets you specify the required parameters for the new node group.

Deckhouse

Search

System Management

Pinned sections

Overview

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

System / System Management / Nodes / Node Groups

Adding a group of type CloudEphemeral

Name\*

name

Required field

Minimum nodes per zone\*

spec.cloudInstances.minPerZone

Number of replicas in MachineDeployment and lower limit in cluster autoscaler

Instance class\*

spec.cloudInstances.classReference.kind

worker

Maximum nodes per zone\*

spec.cloudInstances.maxPerZone

Upper limit in cluster-autoscaler

> • Autoscaler parameters

> • Node Updates

> • Node Template

Create

Page 84



DB

Deckhouse

System / System Management / Nodes / Node Groups

Q Search

System Management

Pinned sections

Overview

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

Instance Classes

Static Instances

Config

Deschedulers

Priority Classes

+ YAML

Create

System / System Management / Nodes / Node Groups

Maximum nodes per zone\*

`spec.cloudInstances.maxPerZone`

Upper limit in cluster-autoscaler

Autoscaler parameters

Zones

`spec.cloudInstances.zones`

Select

Override the zones in which machines are created

Scaling Priority

`spec.cloudInstances.priority`

If there are multiple node groups with the same priority, the group for adding a node will be chosen randomly from them

Number of machines created simultaneously during scale-up

`spec.cloudInstances.maxSurgePerZone`

1

Number of unavailable instances during a RollingUpdate

`spec.cloudInstances.maxUnavailablePerZone`

0

Number of standby nodes from the total number of nodes

`spec.cloudInstances.standby`

Specify a number from the total number of nodes (e.g., 2-6) or a percentage (15%). This is the total number of nodes for all zones at once

Resources occupied on standby nodes, %

`spec.cloudInstances.standbyWolder.overprovisioningRate`

50

Allowed values: 1 <= X <= 80

Quick Shutdown

`spec.cloudInstances.quickShutdown`

Reduces the machine drain waiting time to 5 minutes

☐

> Node Updates

> Node Template

Deckhouse

System Management / System Management / Nodes / Node Groups

Search

System Management

Pinned sections

Overview

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

Instance Classes

Static Instances

Config

Deschedulers

Priority Classes

System / System Management / Nodes / Node Groups

spec.cloudInstances.minPerZone

Number of replicas in MachineDeployment and lower limit in cluster autoscaler

Maximum nodes per zone\*

spec.cloudInstances.maxPerZone

Upper limit in cluster-autoscaler

spec.cloudInstances.classReference.kind

> • Autoscaler parameters

< • Node Updates

Maximum number of simultaneously updated nodes

spec.update.maxConcurrent

You can specify the number of nodes or a percentage of the total number of nodes in this group

1

Reboot Update Mode ⓘ

Auto

Manual

By default

> • Node Template

> • Node System Settings

> • Chaos Monkey parameters

YAML

Create

Deckhouse

Search

System Management

Pinned sections

Overview

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

Instance Classes

Static Instances

Config

Deschedulers

Priority Classes

System / System Management / Nodes / Node Groups

Minimum nodes per zone

spec.cloudInstances.minPerZone

Number of replicas in MachineDeployment and lower limit in cluster autoscaler

Maximum nodes per zone\*

spec.cloudInstances.maxPerZone

Upper limit in cluster-autoscaler

Instance class

spec.cloudInstances.classReference.kind

> Autoscaler parameters

> Node Updates

^ Node Template

Labels

KEY

VALUE

+ ADD

Annotations

KEY

VALUE

+ ADD

Taints

Labels and expressions

Add

> Node System Settings

> Chaos Monkey parameters

+ YAML

Create

Deckhouse

Search

System Management

Pinned sections

Overview

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

Instance Classes

Static Instances

Config

Deschedulers

Priority Classes

System / System Management / Nodes / Node Groups

^ Node System Settings

CRI Settings

ContainerdV2

Containerd

Not managed

By default

Topology Manager

The Topology Manager allows the Kubernetes scheduler to consider resource topology data when scheduling pods.

In a cluster with NUMA nodes, the Topology Manager can improve application performance by placing pod containers on the NUMA node that works more efficiently with resources.

More information in the [Kubernetes documentation](#).

Kubelet Parameters

Maximum Number of Log Files with Rotation Considered

spec.kubelet.containerLogMaxFiles

4

Maximum log file size before rotation

spec.kubelet.containerLogMaxSize

50Mi

Maximum number of pods on a node

spec.kubelet.maxPods

110

Path to Directory for Kubelet Files

spec.kubelet.rootDir

/var/lib/kubelet

GPU Resource Configuration

+ YAML

Create

“Instance classes” subsection

This subsection is intended for managing instance classes used in the cluster. The class list can be sorted.

Page 86

88

Deckhouse

Search

System Management

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

Instance Classes

Static Instances

Config

Deschedulers

Priority Classes

Access

Dex Providers

User Sessions

V

?

+ YAML

System / System Management / Nodes

Instance Classes 4

Create

Search by name

+

| Name         | Image            | Storage Class | VM Class       | Core Fraction | Memory |                                                                                                                                                                         |
|--------------|------------------|---------------|----------------|---------------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| billing-prom | ubuntu-24-04-lts | replicated    | amd-epyc-gen-3 | 100%          | 12Gi   |   |
| frontend     | ubuntu-24-04-lts | replicated    | amd-epyc-gen-3 | 20%           | 4Gi    |   |
| system       | ubuntu-24-04-lts | replicated    | amd-epyc-gen-3 | 20%           | 8Gi    |   |
| worker       | ubuntu-24-04-lts | replicated    | amd-epyc-gen-3 | 20%           | 8Gi    |   |

88

Deckhouse

Search

System Management

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

Instance Classes

Static Instances

Config

Deschedulers

Priority Classes

Access

Dex Providers

User Sessions

V

?

+ YAML

System / System Management / Nodes

Instance Classes 4

Search by name

+

| Name         | Image            |
|--------------|------------------|
| billing-prom | ubuntu-24-04-lts |
| frontend     | ubuntu-24-04-lts |
| system       | ubuntu-24-04-lts |
| worker       | ubuntu-24-04-lts |

worker

Configuration

Meta

YAML

Additional Disks

Root Disk

Image

Kind

ClusterVirtualImage

VirtualImage

VirtualDisk

Name

ubuntu-24-04-lts

Size

50Gi

Storage Class

replicated

Virtual Machine

Bootloader

BIOS

EFI

EFIWithSecureBoot

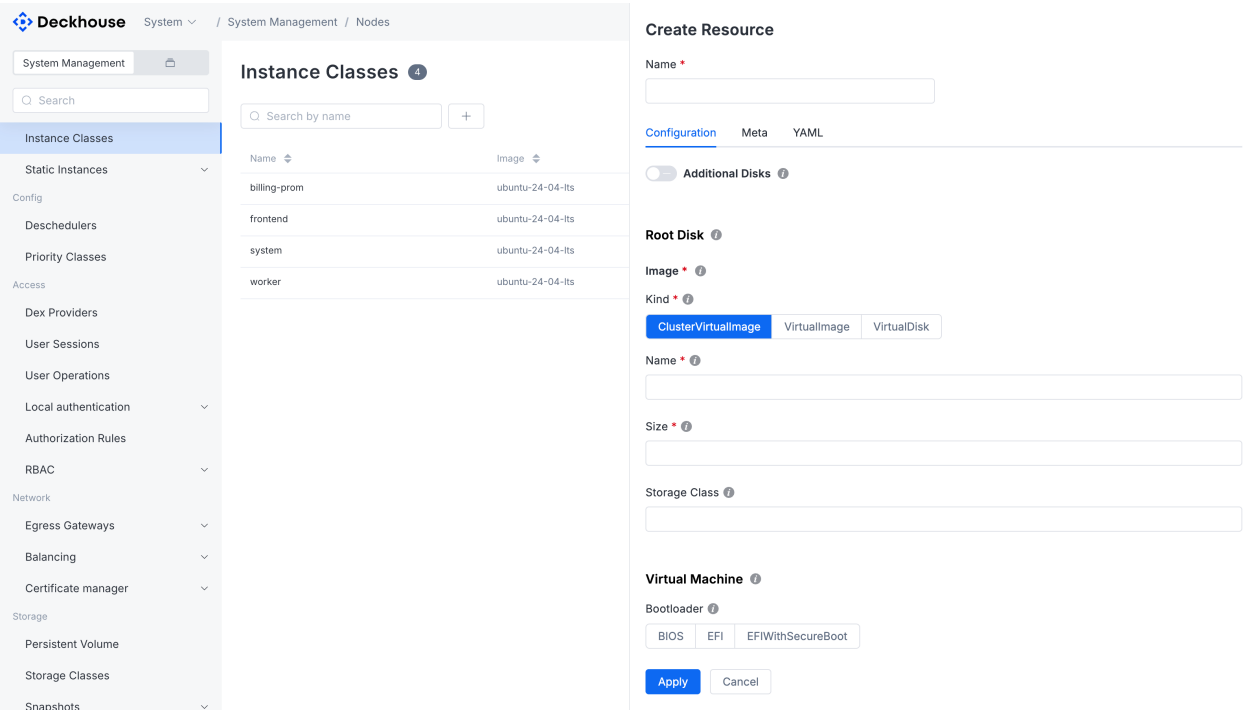
CPU

Core Fraction

Save

Cancel

The “Create resource” form lets you specify parameters for a new instance configuration for the cluster.



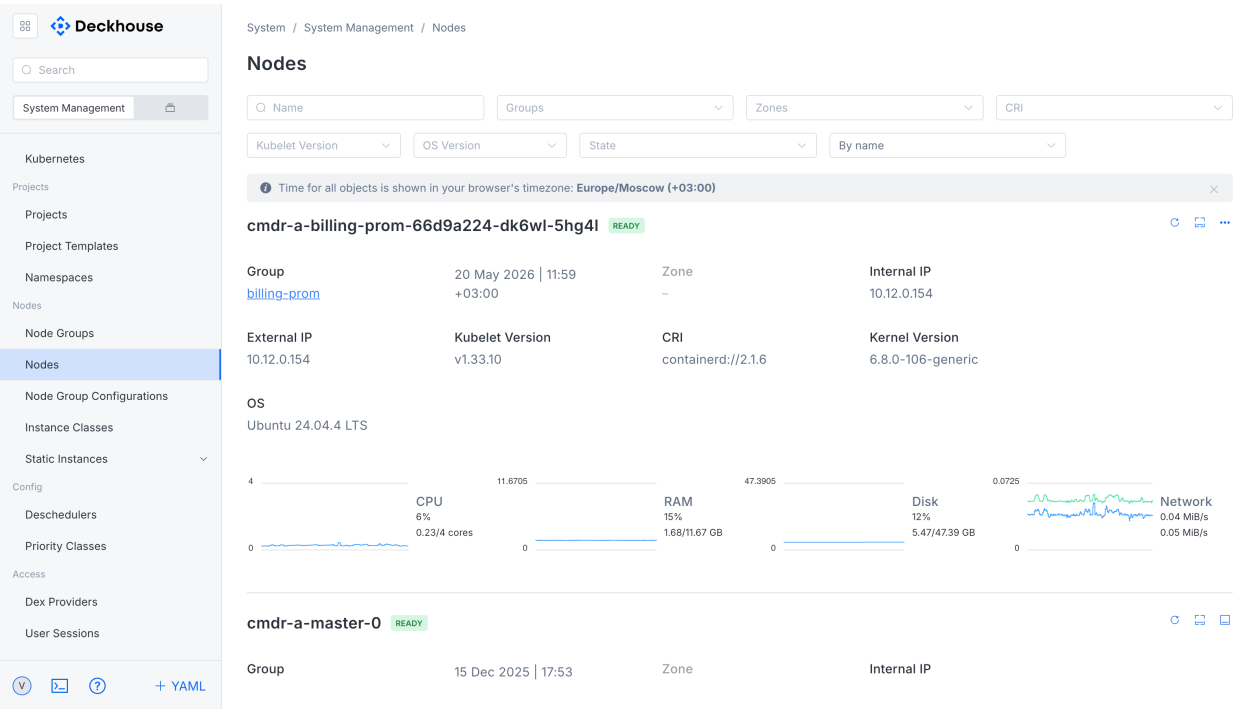
“Nodes” subsection

This subsection provides summary information about all DKP cluster nodes with filtering and sorting capabilities.

The node card displays:

- Current node state
- Group
- Date and time
- Availability zone
- Internal and external IP addresses
- Container runtime in use
- Kernel version
- Kubelet version
- Operating system

The subsection also contains charts showing CPU, memory, disk, and network traffic usage, allowing you to monitor node performance. To manage node availability in the cluster, the “Cordon” and “Cordon+Drain” buttons are available.



## “Static instances” subsection

This subsection is intended for managing static instances in the cluster and includes two tabs:

- “Instances”: For working with static instances.
- “SSH access”: For configuring SSH access.

The interface allows you to quickly find and manage static instances.

The “Add instance” button on the “Instances” tab opens the form for adding a new instance to the cluster. The form requires the following information:

- Instance name
- Instance address
- SSH access method from the drop-down list

Additionally, you can specify labels by defining keys and values for further identification and management of the instance.

The screenshot displays the Deckhouse web interface for creating a new static instance. The left sidebar contains a navigation menu with categories like 'Kubernetes', 'Projects', 'Nodes', 'Config', and 'Access'. The 'Nodes' category is expanded, and 'Instances' is selected. The main panel shows the 'New Static Instance' form with a breadcrumb trail: 'System / System Management / Nodes / Static Instances / Instances'. The form has two tabs: 'Configuration' and 'Meta'. The 'Configuration' tab is active and contains three input fields: 'Name', 'Address', and 'SSH credential'. The 'SSH credential' field is a dropdown menu with 'Select' as the current option. A 'Create' button is located at the bottom right of the form.

The “Add SSH access” button on the “SSH access” tab opens the form for configuring SSH connections to nodes. The form requires the following information:

- SSH access name
- Username
- Private SSH key
- (Optional) `sudo` password for executing privileged commands

Additional fields are available for changing the SSH port and adding extra SSH arguments.

System / System Management / Nodes / Static Instances / SSH Credentials

### New SSH credential

Name \*

Configuration Meta

Username \* Sudo password \* SSH port 22

☐ Private SSH key

SSH extra arguments

Create

## Multitenancy

### “Projects” subsection

This subsection is intended for creating projects based on prepared templates that define the set of resources to be created and their parameters.

During project creation, the following operations are performed:

- Parameter validation against the OpenAPI schema.
- Template rendering through Helm.
- Deployment of all described resources in an automatically created namespace.

Standard Kubernetes mechanisms are used for access control, resource limits, and network isolation. This allows you to manage security and workload inside the namespace.

System / System Management / Projects

### Projects

Create

Search by name +

| Name                 | Status   | Namespaces                                                          | Template        | Description               |
|----------------------|----------|---------------------------------------------------------------------|-----------------|---------------------------|
| commander-project    | DEPLOYED | —                                                                   | commander-basic | —                         |
| commander-project--1 | DEPLOYED | —                                                                   | commander-basic | —                         |
| deckhouse            | DEPLOYED | d8-admission-policy-engine<br>d8-cert-manager<br>d8-chrony          | virtual         | This is a virtual project |
| default              | DEPLOYED | auto-tests-general-namespace-not-deleted<br>cloud-resources-checker | virtual         | This is a virtual project |

The “Create project” button opens the form for adding a new project based on the selected template. In the form, specify the project name and optionally add labels and annotations.

In the central part of the form, select the project template based on which the required resources will be created. If necessary, leave a comment.

The lower part of the form contains fields for entering template parameters and displaying its structure.

System / System Management / Projects / **Projects**

## New project

Name \*

Configuration Meta

Project comment

Project template \*  
default

Resources defined in the [project template](#) will be created for the project.

Configuration YAML

Administrators \* ⓘ

| Name * ⓘ | Subject * ⓘ |  |
|----------|-------------|--|
|          | User        |  |

Add

Cluster Log Destination Name ⓘ

Create

## “Project templates” subsection

This subsection is intended for creating project templates.

Project templates include basic usage scenarios by default and demonstrate DKP capabilities. To add a new template, use the “Create” button in the upper-right corner.

System / System Management / Projects

## Project Templates ⓘ

Create

Search by name

| Name ↕                      | Status | Description ↕                                                                  |  |
|-----------------------------|--------|--------------------------------------------------------------------------------|--|
| 12                          | READY  | —                                                                              |  |
| commander-basic             | READY  | A template that covers basic project use cases.                                |  |
| default                     | READY  | A template that covers basic project use cases.                                |  |
| secure                      | READY  | Includes all the capabilities of the default template and additional features. |  |
| secure-with-dedicated-nodes | READY  | Includes all the capabilities of the secure template and additional features.  |  |

The “New project template” form lets you specify the template name and add labels and annotations for identification.

The form contains two tabs:

- “OpenAPI schema”: For describing the value specification in JSON format.
- “Project resource template”: For defining Helm-compatible resources and managing the project environment.

System / System Management / Projects / **Project Templates**

**default**

Deckhouse

Configuration Meta YAML

openAPI Schema Project resources template

```

1 properties:
2   administrators:
3     description: |
4       Users and groups that will have admin access to the project.
5       Administrators are eligible to manage roles and access to the project.
6     items:
7       properties:
8         name:
9           description: |
10            The name of the target resource to apply access
11            to the environment.
12            minLength: 1
13            type: string
14          subject:
15            description: |
16              Kind of the target resource to apply access to the
17              environment ('Group' or 'User').
18            enum:
19              - User
20              - Group
21            type: string
22          required:
23            - subject
24            - name
25          type: object
26          type: array
27        clusterLogDestinationName:
28          description: >
29            If specified, the project will be monitored by the Deckhouse log shipper
30            and send logs to the specified cluster log destination.

```

Describe the OpenAPI specification for the values (values) used in the described template. The data will be saved to the API in JSON format, so YAML comments will be ignored.

System / System Management / Projects / **Project Templates**

**default**

Deckhouse

Configuration Meta YAML

openAPI Schema Project resources template

```

1 ---
2 apiVersion: v1
3 kind: Namespace
4 metadata:
5   name: {{ .projectName }}
6   labels:
7     {{ with .parameters.podSecurityProfile }}security.deckhouse.io/pod-policy: "{{ lower . }}"{{ end }}
8     {{ if .parameters.extendedMonitoringEnabled }}extended-monitoring.deckhouse.io/enabled: ""{{ end }}
9
10 {{- range $administrator := .parameters.administrators }}
11 ---
12 apiVersion: deckhouse.io/v1alpha1
13 kind: AuthorizationRule
14 metadata:
15   name: {{ $administrator.name | include "normalize" }}
16 spec:
17   accessLevel: Admin
18   subjects:
19     - kind: {{ $administrator.subject }}
20       name: {{ $administrator.name }}
21 {{- end }}
22
23 ---
24 # Max requests and limits for resource and storage consumption for all pods in a namespace.
25 # Refer to https://kubernetes.io/docs/concepts/policy/resource-quotas/
26 apiVersion: v1
27 kind: ResourceQuota
28 metadata:
29   name: all-pods
30 spec:

```

Templates are compatible with all helm functions. Read more about [creating isolated environments](#)

A new template can be created based on an existing one.

System / System Management / Projects / **Project Templates**

**New project template**

Project template name \*

secure

Configuration Meta

openAPI Schema Project resources template

```

1 properties:
2   administrators:
3     description: |
4       Users and groups that will have admin access to the project.
5       Administrators are eligible to manage roles and access to the project.
6     items:
7       properties:
8         name:
9           description: |
10            The name of the target resource to apply access
11            to the environment.
12            minLength: 1
13            type: string
14          subject:
15            description: |
16              Kind of the target resource to apply access to the
17              environment ('Group' or 'User').
18            enum:
19              - User

```

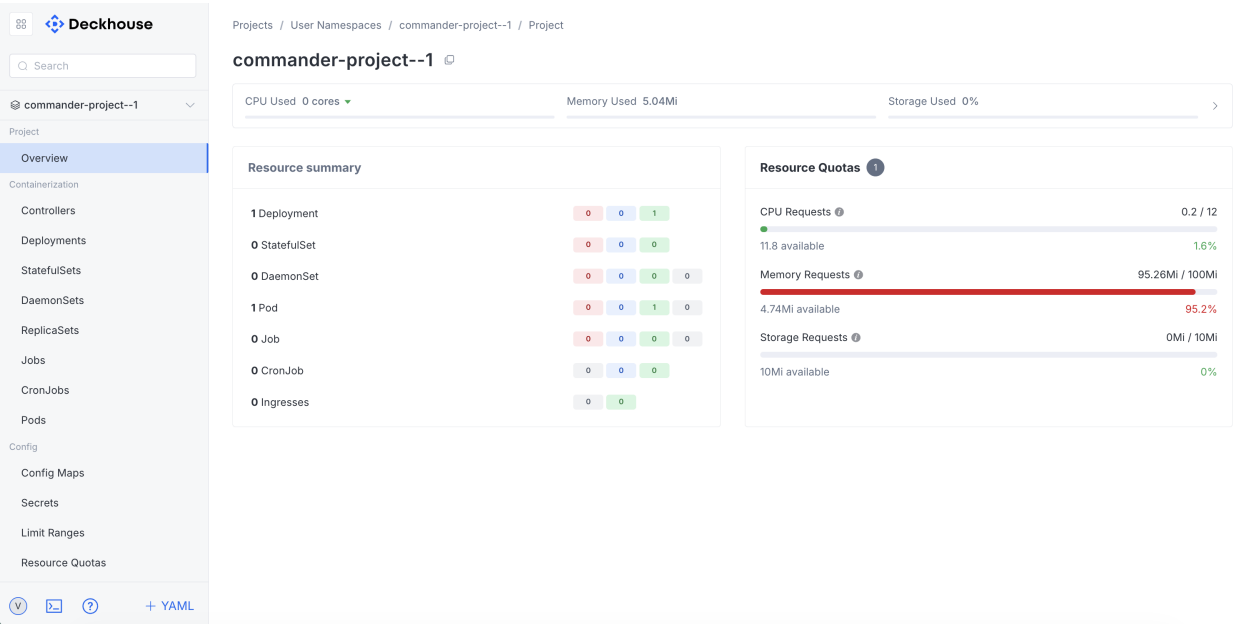
Create



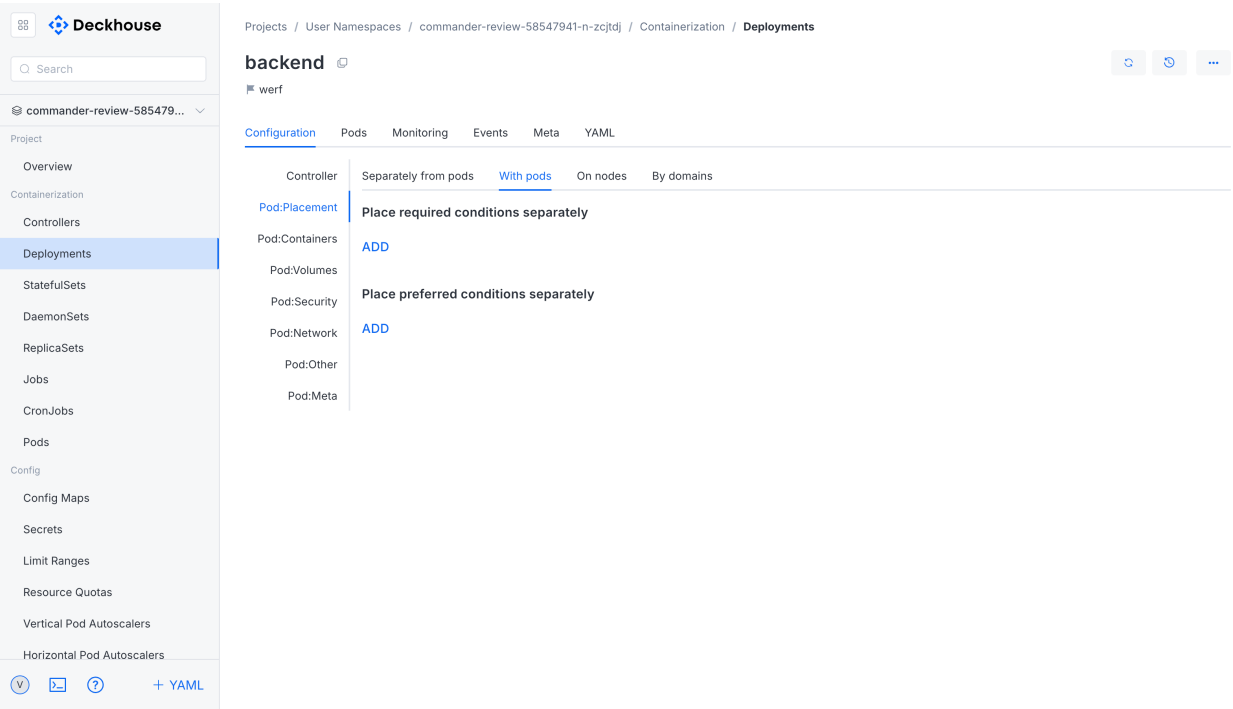
## “User namespaces” subsection

This subsection is intended for managing user project namespaces.

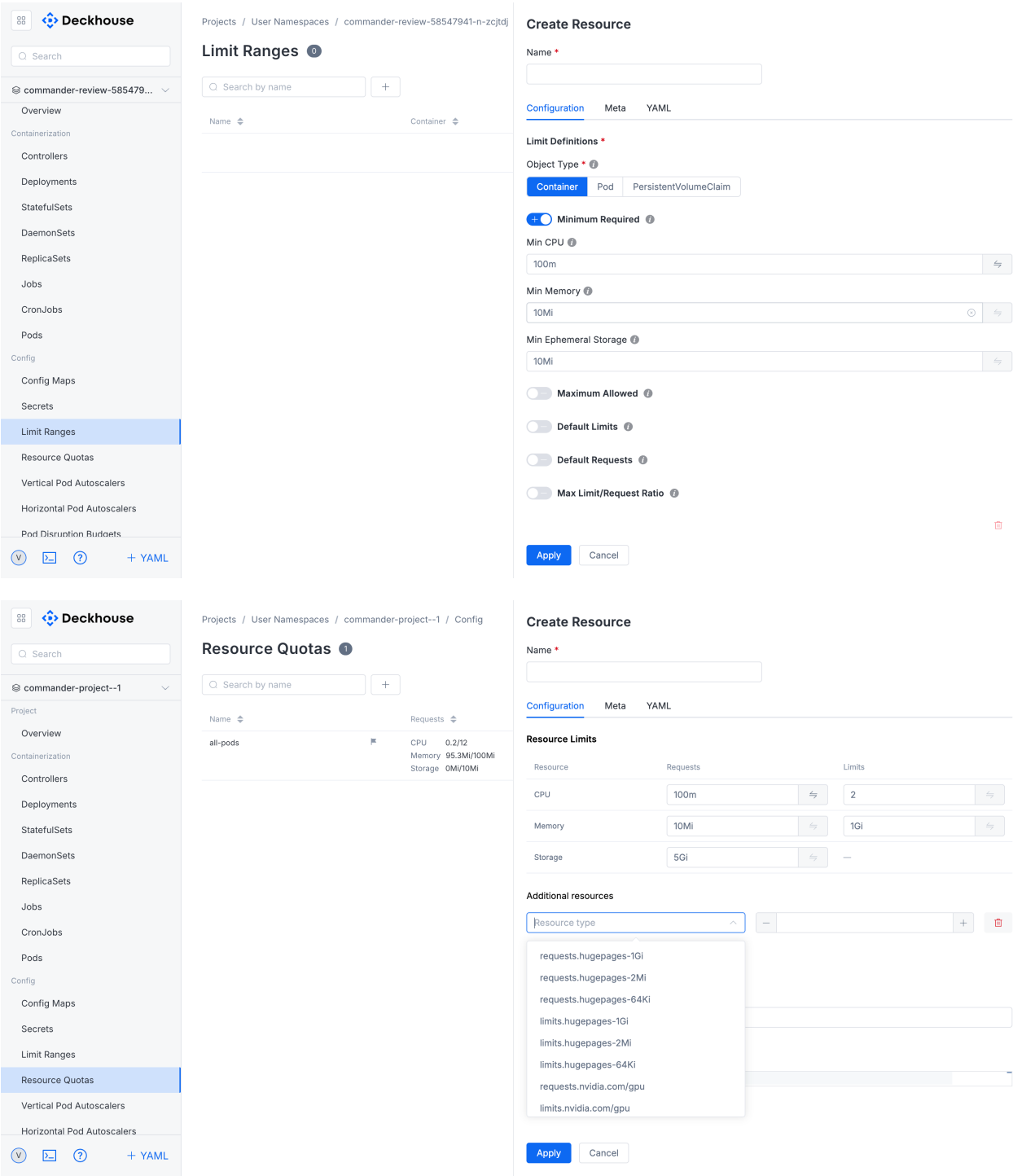
The “Overview” section provides information about the general namespace state.



Here you can manage Deployment parameters and other cluster entities.



You can also configure resource limits and quotas.



# Access

## “User operations” subsection

This subsection provides an interface for user management.

System / System Management / Access

### User operations

Search by email, username or connector All kinds Locked only

| User                                                                         | Status | Locked until | Failed attempts | Actions |
|------------------------------------------------------------------------------|--------|--------------|-----------------|---------|
| <b>zabit@deckhouse.ru</b><br>Local - zabit                                   | ACTIVE | —            | 0               |         |
| <b>yakovlevnv@tt.com</b><br>Local - yakovlev.nikita                          | ACTIVE | —            | 0               |         |
| <b>uservlad@vlad.vlad</b><br>Local - user-vlad                               | ACTIVE | —            | 0               |         |
| <b>user.im@flant.com</b><br>Local - test-user-im-ga                          | ACTIVE | —            | 0               |         |
| <b>user-observability@user.local</b><br>Local - user-observability           | ACTIVE | —            | 0               |         |
| <b>testui@ui.ui</b><br>Local - testedui                                      | ACTIVE | —            | 0               |         |
| <b>testpermission@flant.com</b><br>Local - test-permission-user              | ACTIVE | —            | 0               |         |
| <b>test@test.test</b><br>Local - testT2334                                   | ACTIVE | —            | 0               |         |
| <b>test@test.com</b><br>Local - test-test-com                                | ACTIVE | —            | 0               |         |
| <b>test-superadmin1.user@foxtrot-dev.ru</b><br>Local - test-user-superadmin1 | ACTIVE | —            | 0               |         |
| <b>test-superadmin.user@foxtrot-dev.ru</b><br>Local - test-user-superadmin   | ACTIVE | —            | 0               |         |
| <b>test-poli@mail.com</b><br>Local - test-poli                               | ACTIVE | —            | 0               |         |

If necessary, you can block a user or reset a password.

System / System Management / Access

### User operations

Search by email, username or connector All kinds Locked only

| User                                                                         | Status | Locked until | Failed attempts | Actions |
|------------------------------------------------------------------------------|--------|--------------|-----------------|---------|
| <b>zabit@deckhouse.ru</b><br>Local - zabit                                   | ACTIVE | —            | 0               |         |
| <b>yakovlevnv@tt.com</b><br>Local - yakovlev.nikita                          | ACTIVE | —            | 0               |         |
| <b>uservlad@vlad.vlad</b><br>Local - user-vlad                               | ACTIVE | —            | 0               |         |
| <b>user.im@flant.com</b><br>Local - test-user-im-ga                          | ACTIVE | —            | 0               |         |
| <b>user-observability@user.local</b><br>Local - user-observability           | ACTIVE | —            | 0               |         |
| <b>testui@ui.ui</b><br>Local - testedui                                      | ACTIVE | —            | 0               |         |
| <b>testpermission@flant.com</b><br>Local - test-permission-user              | ACTIVE | —            | 0               |         |
| <b>test@test.test</b><br>Local - testT2334                                   | ACTIVE | —            | 0               |         |
| <b>test@test.com</b><br>Local - test-test-com                                | ACTIVE | —            | 0               |         |
| <b>test-superadmin1.user@foxtrot-dev.ru</b><br>Local - test-user-superadmin1 | ACTIVE | —            | 0               |         |
| <b>test-superadmin.user@foxtrot-dev.ru</b><br>Local - test-user-superadmin   | ACTIVE | —            | 0               |         |
| <b>test-poli@mail.com</b><br>Local - test-poli                               | ACTIVE | —            | 0               |         |

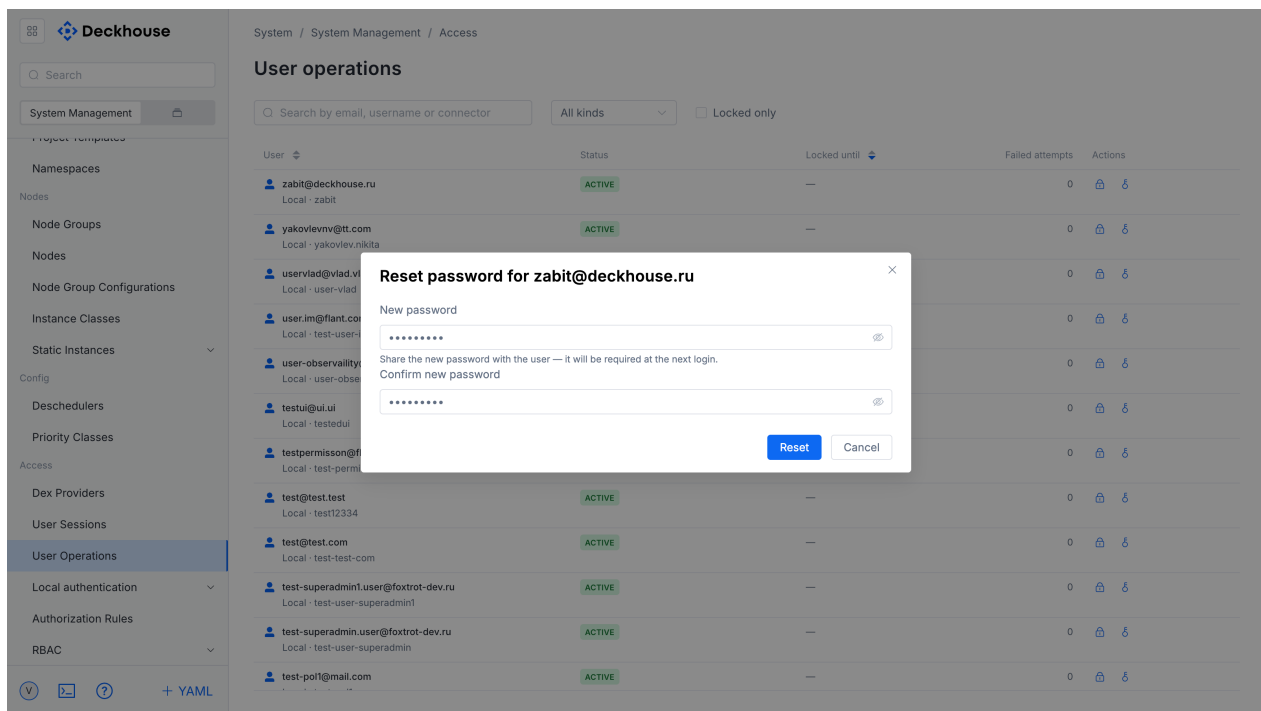
**Lock zabit@deckhouse.ru**

Duration

1h

Format examples: 30s, 10m, 1h, 2h30m, 7d.

**Lock** **Cancel**



## Network

### “Ingress controllers” subsection

This subsection displays information about the current Ingress controllers responsible for traffic routing inside the DKP cluster. If necessary, you can sort the list of controllers and add new ones.

The “Nginx Ingress controllers” card displays its main parameters:

- Inbound connection type (LoadBalancer)
- IP address and Ingress class ( `nginx` )
- Load balancer access level
- Node selector that determines on which nodes the controller runs

The lower part of the card contains monitoring charts for CPU usage, memory usage, network traffic, and requests per second (RPS), allowing you to monitor controller performance. The “Create from” option is available for cloning the configuration, and the “Delete” option is available for removing the Ingress controller.

The “Add” button in the upper-right corner allows you to select the type of a new inbound connection (inlet) for the Ingress controller. Available options:

- HostPort
- HostPort with Proxy Protocol
- HostPort with a failover controller
- LoadBalancer
- LoadBalancer with Proxy Protocol

Deckhouse

in

System Management

Overview

Modules

Deckhouse

Global settings

Nodes

Instance Classes

Static Instances

Instances

Access

RBAC

Cluster Role Bindings

Roles Bindings

Access in nodes

Network

Balancing

Ingress Controllers

Security

V

D

?

+ YAML

System / System Management / Network / Balancing

### Ngix Ingress Controllers

Ingress Controller Types

By name

HostPort  
nginx

Unrestricted

Port HTTP / HTTPS  
80 / -

CPU  
0.2235  
0.03 cores

RAM  
1.3352  
0.91 GB

Network  
0.0164  
0.01 MiB/s

RPS  
1  
op/s

nginx

Inlet LoadBalancer

Ingress class nginx

Version -

Access to the load balancer Unrestricted

IP 185.11.73.177

Hostname -

Node selector

node-role.kubernetes.io/master:

CPU  
0.5907  
0.07 cores

RAM  
1.4786  
0.66 GB

Network  
4.1372  
0.55 MiB/s

RPS  
111.4640  
3.53 op/s

Storage

“Storage classes” subsection

This subsection is intended for managing storage classes.

Deckhouse

Search

System Management

Access in nodes

Network

Egress Gateways

Gateways

Policies

Balancing

Ingress Controllers

Certificate manager

Storage

Persistent Volume

Storage Classes

Snapshots

Security

Vulnerability Reports

Compliance Reports

Security Policies

Operation Policies

V

D

?

+ YAML

System / System Management / Storage

### Storage Classes

Search

| Name       | Type               | Reclaim Policy | Volume Binding       | PV Expansion |  |
|------------|--------------------|----------------|----------------------|--------------|--|
| local      | Base Storage Class | Delete         | WaitForFirstConsumer | Allowed      |  |
| replicated | Base Storage Class | Delete         | WaitForFirstConsumer | Allowed      |  |

“Persistent volume” subsection

This subsection is intended for managing persistent volumes.

88

Deckhouse

Search

System Management

Access in nodes

Network

Egress Gateways

Gateways

Policies

Balancing

Ingress Controllers

Certificate manager

Storage

Persistent Volume

Storage Classes

Snapshots

Security

Vulnerability Reports

Compliance Reports

Security Policies

Operation Policies

V

?

+ YAML

System / System Management / Storage

Persistent Volume103

Create

Storage12.9%

Search by name

| Name                                     | Status   | Storage | Access Modes  | Storage Class | PVC                                  |
|------------------------------------------|----------|---------|---------------|---------------|--------------------------------------|
| pvc-0667bb35-e4f2-4d6b-9b28-f6b1e308d0d4 | RELEASED | 20Gi    | ReadWriteOnce | replicated    | data-billing-prometheus-0            |
| pvc-06ecce60-bbcc-419b-817d-dfd5ec361b73 | RELEASED | 1Gi     | ReadWriteOnce | replicated    | gc-pv-test-pvc                       |
| pvc-0d7e6add-8b4e-4688-b07a-2595cea4e483 | RELEASED | 1Gi     | ReadWriteOnce | local         | d8ms-pg-test-1                       |
| pvc-0f15edc2-6cbe-4561-9889-1876af5eeee3 | BOUND    | 20Gi    | ReadWriteOnce | replicated    | data-billing-prometheus-1            |
| pvc-105aab22-2fee-478d-8eae-41814e90c4e2 | RELEASED | 2Gi     | ReadWriteOnce | replicated    | data-billing-reports-0               |
| pvc-1131ec48-5554-470d-abee-6053f8aa5995 | BOUND    | 2Gi     | ReadWriteOnce | replicated    | data-commander-postgres-0            |
| pvc-1fd0b7c-da25-4b27-9e2e-a5ac46b23e16  | BOUND    | 20Gi    | ReadWriteOnce | replicated    | data-billing-prometheus-0            |
| pvc-14e3f3fc-249b-419d-8760-a0196ffa3c28 | BOUND    | 20Gi    | ReadWriteOnce | replicated    | data-billing-prometheus-1            |
| pvc-19f9a2a0-ea46-4bd2-9d01-ec731c4969df | BOUND    | 2Gi     | ReadWriteOnce | replicated    | data-commander-postgres-0            |
| pvc-1be8e7b7-0610-4614-9fcb-096f87a355cd | BOUND    | 40Gi    | ReadWriteOnce | replicated    | prometheus-main-db-prometheus-main-1 |

“Local volumes” subsection

This subsection is intended for managing local volumes. It contains four sections for managing pools, volume groups, logical volumes, and block devices.

88

Deckhouse

Search

System Management

Priority Classes

Access

Dex Providers

User Sessions

User Operations

Local authentication

Authorization Rules

RBAC

Network

Egress Gateways

Balancing

SDN

Certificate manager

Storage

Persistent Volume

Storage Classes

Snapshots

LVM

V

?

+ YAML

System / System Management / Storage / LVM

Pools

Create

Search

| Name      | Status | Type    |
|-----------|--------|---------|
| 111111    | FAILED | LVM     |
| 12345     | FAILED | LVM     |
| 123451    | FAILED | LVM     |
| 123466    | FAILED | LVMThin |
| thin-data | FAILED | LVMThin |

88

Deckhouse

Q Search

System Management

Authorization Rules

RBAC

Network

Egress Gateways

Balancing

SDN

Certificate manager

Storage

Persistent Volume

Storage Classes

Snapshots

LVM

Pools

Volume Groups

Logical Volumes

Block Devices

Security

Audit Rules

V

?

+ YAML

System / System Management / Storage / LVM

Volume Groups

Create

| Name                                    | Status                                                   | Node           | VG on the node | Thin pools | Size  | Allocated size | Age      |                         |
|-----------------------------------------|----------------------------------------------------------|----------------|----------------|------------|-------|----------------|----------|-------------------------|
| vg-053f1be4-32c5-46e3-934c-38d4520c9f9e | <div>ERROR</div> <div>CONFIGURATION IS NOT APPLIED</div> | d8-virt-node-2 | vg-sds         | 0/0        | 20Gi  | 1Gi            | 7 months | <div></div> <div></div> |
| vg-0ac5c7b6-1017-4301-b7c3-e9bdd410071d | <div>ERROR</div> <div>CONFIGURATION IS NOT APPLIED</div> | d8-virt-node-1 | vg-sds         | 0/0        | 20Gi  | 0              | 7 months | <div></div> <div></div> |
| vg-110158a1-bb3c-4d90-a7be-44b3f472532  | <div>ERROR</div> <div>CONFIGURATION IS NOT APPLIED</div> | d8-virt-node-2 | vg             | 1/1        | 10Gi  | 10Gi           | 4 months | <div></div> <div></div> |
| vg-13642112-5225-43fc-acf8-5615634195ed | <div>ERROR</div> <div>CONFIGURATION IS NOT APPLIED</div> | d8-virt-node-0 | vg-1           | 1/1        | 30Gi  | 30Gi           | 5 months | <div></div> <div></div> |
| vg-15281210-3f8e-437c-8a41-b9b67d6d95d2 | <div>ERROR</div> <div>CONFIGURATION IS NOT APPLIED</div> | d8-virt-node-2 | vg             | 1/1        | 10Gi  | 10Gi           | 2 months | <div></div> <div></div> |
| vg-1c06db7d-50b3-45bd-b333-44e313f8f8fb | <div>ERROR</div> <div>CONFIGURATION IS NOT APPLIED</div> | d8-virt-node-0 | vg             | 1/1        | 10Gi  | 10Gi           | 4 months | <div></div> <div></div> |
| vg-24149a46-b5f5-47c3-a71e-274b28142778 | <div>ERROR</div> <div>CONFIGURATION IS NOT APPLIED</div> | d8-virt-node-3 | vg-1           | 1/1        | 30Gi  | 30Gi           | 6 months | <div></div> <div></div> |
| vg-37a218ec-6c51-40d4-aa1c-f78e2bcae020 | <div>ERROR</div> <div>CONFIGURATION IS NOT APPLIED</div> | d8-virt-node-2 | vg-1           | 1/1        | 30Gi  | 30Gi           | 6 months | <div></div> <div></div> |
| vg-3bcbeff53-804c-4404-a20-fb9c004ba2ff | <div>ERROR</div> <div>CONFIGURATION IS NOT APPLIED</div> | d8-virt-node-4 | vg             | 1/1        | 10Gi  | 10Gi           | 4 months | <div></div> <div></div> |
| vg-3facff42-7c29-4543-9024-c5d562f8bdce | <div>ERROR</div> <div>CONFIGURATION IS NOT APPLIED</div> | d8-virt-node-2 | vg-replicated  | 0/0        | 233Gi | 0              | 8 months | <div></div> <div></div> |
| vg-40ed58dc-2619-463a-9258-fd873d317ebd | <div>ERROR</div> <div>CONFIGURATION IS NOT APPLIED</div> | d8-virt-node-2 | vg             | 1/1        | 10Gi  | 10Gi           | 2 months | <div></div> <div></div> |

88

Deckhouse

Q Search

System Management

Authorization Rules

RBAC

Network

Egress Gateways

Balancing

SDN

Certificate manager

Storage

Persistent Volume

Storage Classes

Snapshots

LVM

Pools

Volume Groups

Logical Volumes

Block Devices

Security

Audit Rules

V

?

+ YAML

System / System Management / Storage / LVM

Logical Volumes

| Name                                     | Status  | Volume Group                          | Type | Size  |
|------------------------------------------|---------|---------------------------------------|------|-------|
| pvc-060378b0-e130-4319-b68b-f2f912448bd5 | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-085934ba-7920-4b7f-9a96-dde8e4cde0f7 | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-105569c2-df32-4412-9461-9ae364eac4f1 | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 50Gi  |
| pvc-1a559f4b-5554-46fd-9287-088a518c3ac0 | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-1a56ec0a-df77-497b-950a-21de3915310b | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-1e00f866-bdad-4c34-97d9-4da2bd1dd1be | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-2aedb539-60fe-4bae-85e6-11f8f9bb2e4a | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-2e8274ad-5d08-4d50-8533-6398444121fd | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-2fb15b45-f08e-421f-8471-4657f755edae | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-32469aa4-48a5-4cf8-af22-4365cb43dba3 | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-362276e4-25d7-47fb-929b-f4a15b301c82 | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-3dcc76d4-3397-4ee6-8f19-860dacbc2fe0 | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-3e76b63f-82c4-4008-8477-7792ff065d8f | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-41326f0d-98c9-4514-81ef-219b6f008ebb | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-41abf4ad-3124-4bb3-95c7-dacc2560c06e | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-4216b64f-6d23-4cab-8dd6-5b76f89453f2 | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 15Gi  |
| pvc-44ee680d-396f-4509-8163-d756398ee91f | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |
| pvc-476c4cac-bf52-44c8-a5bb-5938c5634d35 | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 50Gi  |
| pvc-48e7f64e-3c27-4961-8ed2-b748fe9fd9e  | CREATED | vg-sds-local-d8-virt-node-3-nvme4nlp2 | Thin | 200Mi |

88

Deckhouse

Search

System Management

Authorization Rules

RBAC

Network

Egress Gateways

Balancing

SDN

Certificate manager

Storage

Persistent Volume

Storage Classes

Snapshots

LVM

Pools

Volume Groups

Logical Volumes

Block Devices

Security

Audit Rules

V

?

+ YAML

System / System Management / Storage / LVM

Block Devices20

Search by name

| Name                                              | Status   | Node           | Volume group                          | Path           | Size     | Age    |
|---------------------------------------------------|----------|----------------|---------------------------------------|----------------|----------|--------|
| dev-0e8f4f371b9ab33158e9899b1e<br>c629e198c407bb  | OCCUPIED | d8-virt-node-0 | vg-data-d8-virt-node-0-nvme4n1p1      | /dev/nvme4n1p1 | 915709Mi | a year |
| dev-17f9a05b8ff85ef37f06ca0cd4<br>c6260de9dc494   | OCCUPIED | d8-virt-node-3 | vg-sds-local-d8-virt-node-3-nvme4n1p2 | /dev/nvme4n1p2 | 915710Mi | a year |
| dev-30bc9a2dff5ebbbb09d930f7b<br>cad04b6af48a73   | OCCUPIED | d8-virt-node-0 | vg-sds-local-d8-virt-node-0-nvme4n1p2 | /dev/nvme4n1p2 | 915710Mi | a year |
| dev-32449d1007c9656a5aa1721055<br>3307b93422b3e4  | OCCUPIED | d8-virt-node-1 | vg-data-d8-virt-node-1-nvme4n1p1      | /dev/nvme4n1p1 | 915709Mi | a year |
| dev-4c9edd8f541955c823496d849<br>1283caf1ea9b772  | OCCUPIED | d8-virt-node-4 | vg-data-d8-virt-node-4-nvme3n1p1      | /dev/nvme3n1p1 | 915709Mi | a year |
| dev-50415ebc3d5c10e33d62705ed<br>c0ea6b5da7a9a7d  | OCCUPIED | d8-virt-node-2 | vg-data-d8-virt-node-2-nvme4n1p1      | /dev/nvme4n1p1 | 915709Mi | a year |
| dev-6602ecf5b30077712c90047ca1<br>665a19d3accb27  | OCCUPIED | d8-virt-node-3 | vg-sds-local-d8-virt-node-3-nvme4n1p2 | /dev/nvme4n1p2 | 915710Mi | a year |
| dev-6935a2074a390df790b5f6cbe<br>cdc0eeb840aaba4  | OCCUPIED | d8-virt-node-3 | vg-data-d8-virt-node-3-nvme4n1p1      | /dev/nvme4n1p1 | 915709Mi | a year |
| dev-75fcfab96829a449320562e95<br>108b1b5d0c266a   | OCCUPIED | d8-virt-node-1 | vg-data-d8-virt-node-1-nvme4n1p1      | /dev/nvme4n1p1 | 915709Mi | a year |
| dev-850de5d6f65dbac17d3cd8f16e<br>85dd0332cc4770  | OCCUPIED | d8-virt-node-1 | vg-sds-local-d8-virt-node-1-nvme4n1p2 | /dev/nvme4n1p2 | 915710Mi | a year |
| dev-a5945f6845e840f5f8013c43f<br>3e48645f12eb88   | OCCUPIED | d8-virt-node-2 | vg-data-d8-virt-node-2-nvme4n1p1      | /dev/nvme4n1p1 | 915709Mi | a year |
| dev-abe61ea03b1a8bc9b7e6f4428<br>5a2a2a2a2a2a2a2a | OCCUPIED | d8-virt-node-3 | vg-data-d8-virt-node-3-nvme4n1p1      | /dev/nvme4n1p1 | 915709Mi | a year |

Security

“CVE scanner” subsection

This subsection is intended for scanning container images for vulnerabilities (CVEs) in the DKP cluster.

The subsection contains the “Vulnerability reports” and “Scanned namespaces” tabs:

- “Vulnerability reports” tab displays the results of the latest scans. It provides information about the scanned object, including its name, namespace, type, and resource name, as well as the container and the image in use. If no vulnerabilities are found, a green indicator is displayed. The “Rescan” button allows you to perform a new scan.
- “Scanned namespaces” tab is intended for managing namespaces that are subject to scanning. The interface supports sorting by name and scan parameters. You can optionally hide system namespaces. The required namespaces can be selected for scanning, and scans can be started using the “Rescan” button. Reports for each object are also available for viewing.

88

Deckhouse

Search

System Management

Vulnerability Reports

Compliance Reports

Security Policies

Operation Policies

Other reports

Config Audit

RBAC Assessment

Cluster RBAC Assessment

Cluster Infra Assessment

Virtualization

Configuration

Monitoring

Explore data

Dashboards

Active alerts

Alert settings

Grafana Datasources

V

?

+ YAML

System / System Management / Security

CVE Scanner

Vulnerability ReportsScannable Namespaces

Namespace (exact match)

Time for all objects is shown in your browser's timezone: Europe/Moscow (+03:00)

replicaset-8fbb55d6f

Critical1

High20

Medium14

Low2

Unknown0

SCANNED

an hour

NAMESPACE

console-docs-main

KIND/NAME

ReplicaSet/console-docs-dex-authenticator-5d9957dd5f

CONTAINER

dex-authenticator

IMAGE

registry.deckhouse.ru/deckhouse/ee@sha256:14bf1d472ece32089f56acca02bbd809d1847ecd3bda7bab064672a25781cebe

COMPONENTS

github.com/go-jose/go-jose/v3

github.com/go-jose/go-jose/v4

google.golang.org/grpc

stdlib

replicaset-b854467

Critical

High

Medium

Low

Unknown

Monitoring

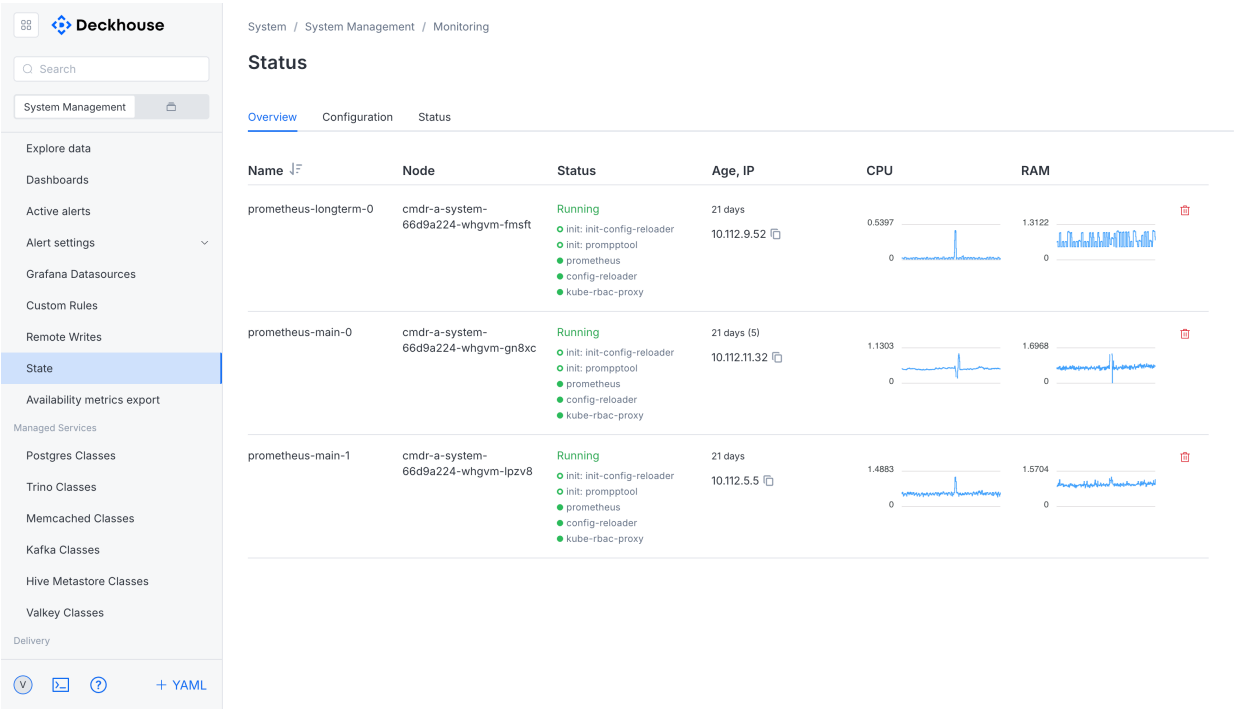
“Overview” subsection



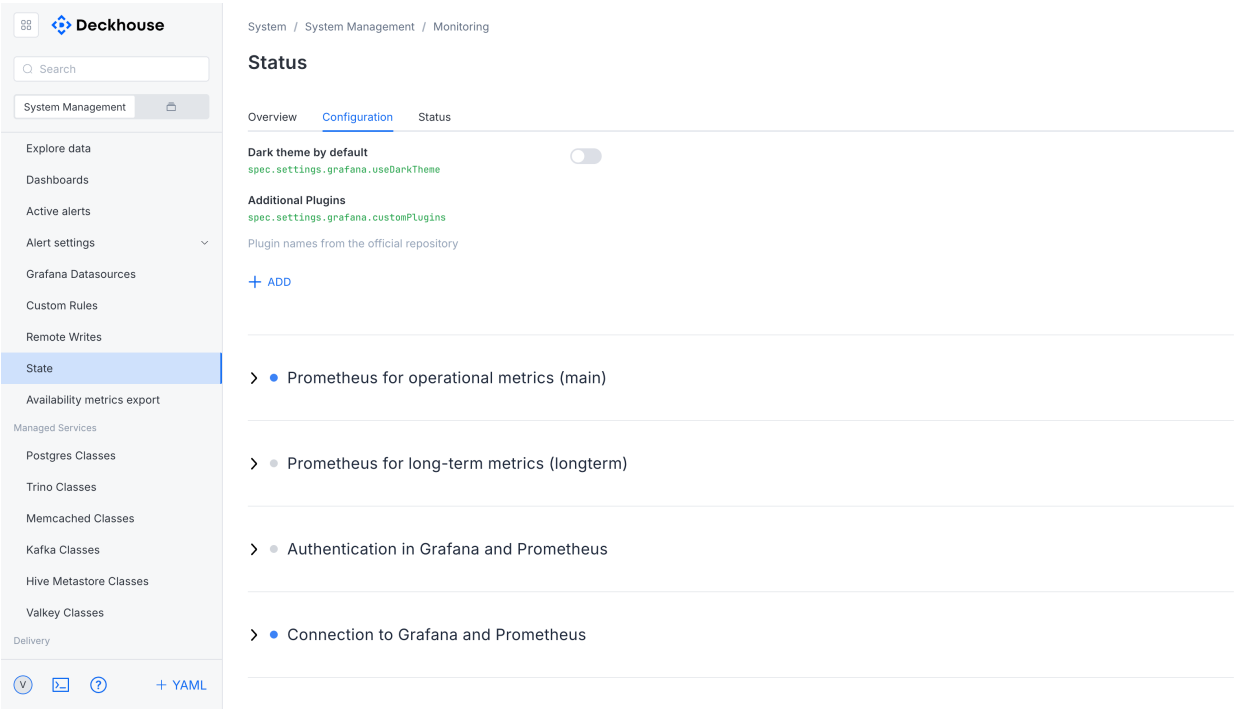
This subsection displays the state of Prometheus instances, load metrics, and a section with settings.

The “Overview” subsection contains the “Status” and “Configuration” tabs intended for monitoring and configuring Prometheus instances.

The “Status” tab displays a list of running pods with their names, assigned nodes, statuses, IP addresses, ages, and CPU and memory usage. Each pod includes components such as `init-config-reloader`, `prometheus`, `config-reloader`, and `kube-rbac-proxy`, which ensure its operation. The button on the right allows you to delete the selected pod.



The “Configuration” tab contains expandable sections for configuring various aspects of Prometheus operation, including real-time and retrospective metrics, authentication and Grafana integration, as well as resource management. This interface is intended for monitoring metric states in real time and for flexible integration with other services.



## “Metric processing” subsection

This subsection allows you to create and manage metric processing rules. When adding a new rule, specify its name and processing group. This makes it possible to organize and modify incoming metrics before they are forwarded further.

System / System Management / Monitoring / Custom Rules

## New metrics rule

Name\*  
metadata.name

+ Add group

Create

## “Metric delivery” subsection

This subsection is intended for configuring data export to a local or external Prometheus server. When adding a new resource, specify the URLs for metric delivery, and configure TLS parameters, authentication, and optional metric preprocessing before delivery.

System / System Management / Monitoring / Remote Writes

## A resource for including data from the local Prometheus

Name\*  
metadata.name

### Configuration

URL for sending metrics\*  
spec.url

> • TLS Parameters ☐

> • Authentication

Metric processing before sending  
spec.writeRelabelConfigs  
Allows you to delete metrics or replace labels.  
More details on configuration in the [Prometheus documentation](#)

Create

## “Grafana data sources” subsection

In this subsection, you can configure Grafana integration with various data sources used in dashboards and visualize data.

To create a new data source, specify its name, type, URL, access parameters, and authentication settings.

Deckhouse

Search

System Management

Active alerts

Alert settings

Grafana Datasources

Custom Rules

Remote Writes

State

Availability metrics export

Managed Services

Postgres Classes

Trino Classes

Memcached Classes

Kafka Classes

Hive Metastore Classes

Valkey Classes

Delivery

Helm Addons

Helm Addon Charts

Helm Addon Repositories

Logging

Log Destinations

Logging Configs

Other

Other CRD

System / System Management / Monitoring / Grafana Datasources

Add a new data source for Grafana

Source Name \*  
metadata.name

Datasource Type \*  
spec.type

The resource parameters are described in detail in the [Grafana documentation](#)

Find the type in the [official documentation](#) or in the [Grafana plugin catalog](#)

URL  
spec.url

Data Access \*  
spec.access

Server (proxy)

Parameters for jsonData  
spec.jsonData

Create

## “Grafana dashboards” subsection

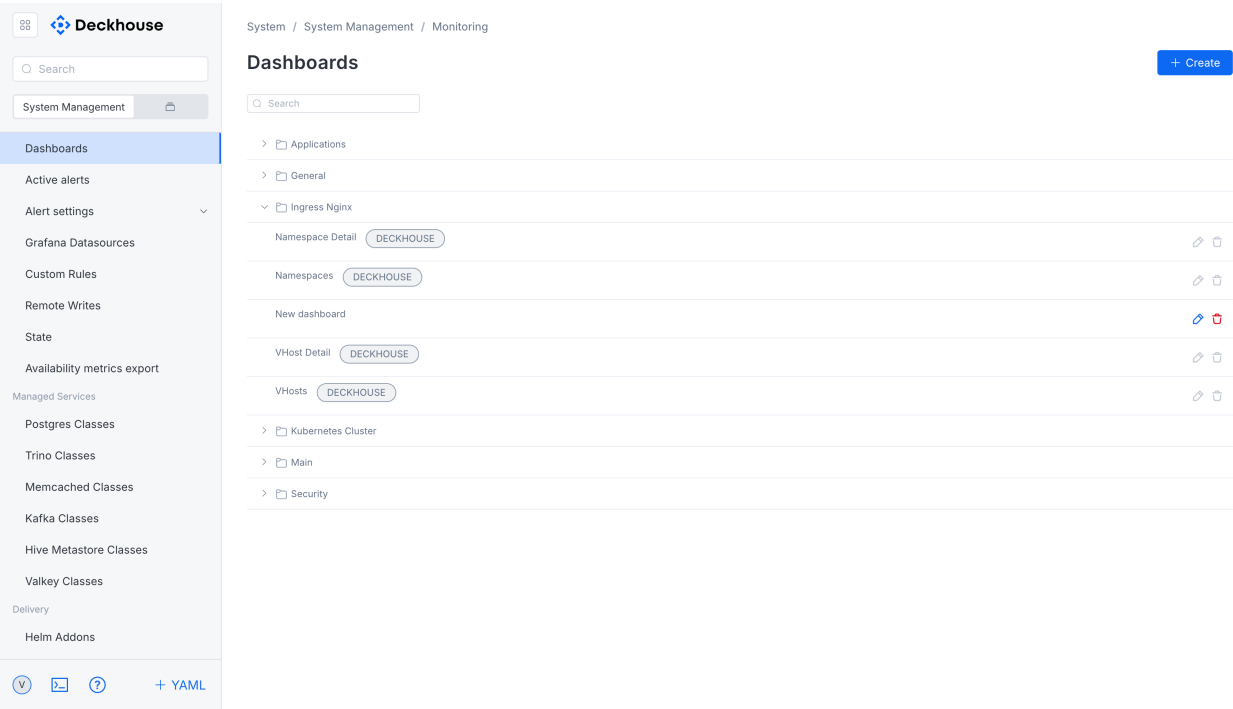
This subsection is intended for managing dashboards used for metric visualization.

The main interface displays a list of available dashboards. You can sort by creation time and filter by name or folder.

The dashboard row displays its name, associated folder, and the “Create from” option for creating a new dashboard based on it, as well as the “Delete” option.

To add a new dashboard:

1. Click the “Add” button.
2. Specify the new dashboard name and the folder in which it will be stored (if the folder does not exist, it will be created automatically).
3. Enter a JSON manifest containing the dashboard configuration description. It is important that the manifest does not contain a local ID (except for the UID), otherwise the dashboard may not be displayed correctly in Grafana.

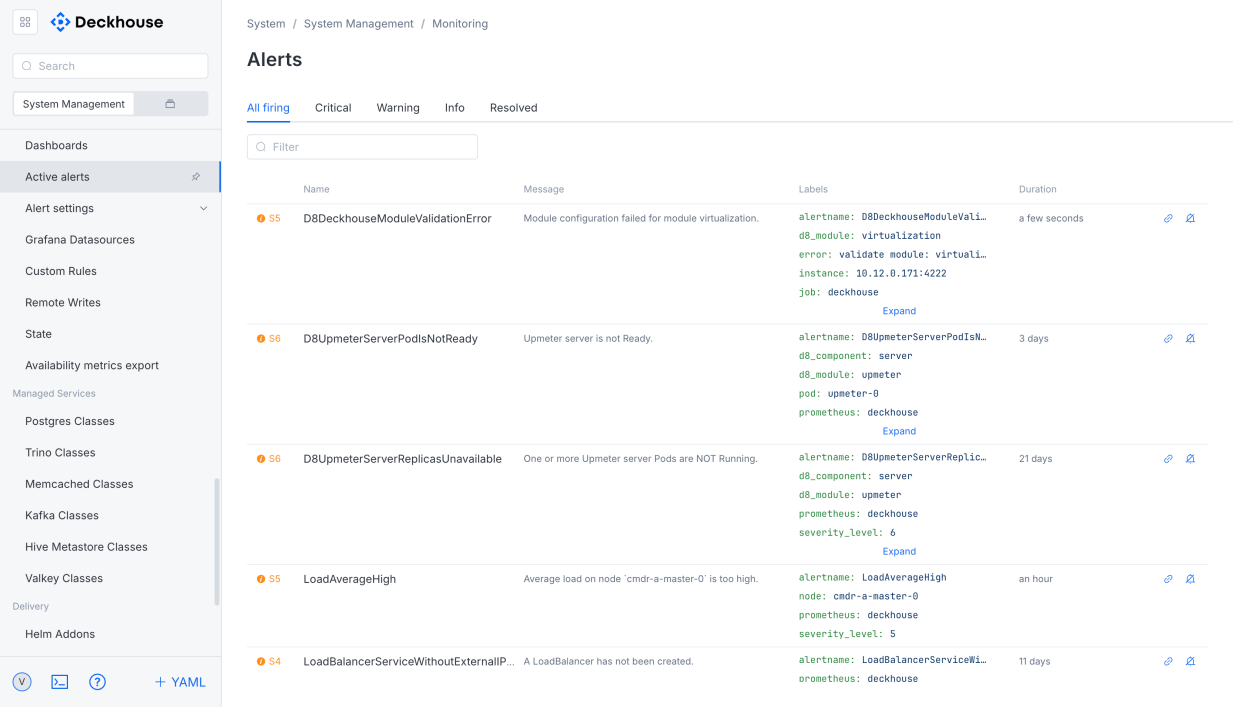


“Active alerts” subsection

This subsection displays a list of current monitoring system alerts. Alerts can be sorted by name and status.

The alert row displays the alert name, severity level, creation time, and last update time. Related components and modules are also displayed to help identify the source of the issue.

To get detailed information about an alert, click “Read description”. At the bottom, an explanation of the alert trigger reason is displayed.



A list of all monitoring system alerts available in DKP is provided on a separate documentation page.

Logging

This section is intended for configuring log sources and managing log delivery.

“Log delivery” subsection

This subsection is intended for managing logging and configuring log delivery to various storage systems.

To configure log delivery:

1. Click the “Add” button.
2. Select one of the target storage systems:
  - Loki
  - ElasticSearch
  - Logstash
  - Vector
  - Kafka
  - Splunk
3. Specify the storage system name and connection address.
4. If necessary, specify additional settings:
  - TLS parameters for a secure connection.
  - Authentication (“Basic” or “Bearer token”).
  - Additional labels for filtering and organizing logs.
  - Buffer parameters that define how logs are stored before delivery (on disk or in memory).
  - Delivery limits that allow you to configure the log record delivery rate.
  - Exclusions that allow you to filter specific logs.

System / System Management / Logging / Log Destinations

### Add new log storage – Loki

Type: Loki  
spec.type

Name\*  
metadata.name

Connection parameters

Connection

Loki address\*  
spec.Loki.endpoint

The agent automatically adds /loki/api/v1/push to the URL when sending data

> • TLS parameters ☐

Authentication  
spec.Loki.auth.strategy

☒ No ☐ Basic ☐ Bearer token

Additional labels ⓘ  
spec.extraLabels

Labels will be added to the logs. You can use simple templates using [Vector template syntax](#), for example {{ app }}

| KEY   | VALUE |
|-------|-------|
| + ADD |       |

Create

## “Log collection” subsection

This subsection is intended for configuring log sources that can then be delivered to target storage systems. You can sort and filter existing log collection rules and add new sources.

To configure collection sources:

1. Click the “Add” button.
2. Select one of the sources:
  - File (collecting logs from the file system)
  - KubernetesPods (collecting logs from Kubernetes pods)
3. Specify the name and configure the parameters:
  - “Storage”: Specify where the collected logs will be delivered.
  - “File filters”: Specify the paths to log files that should or should not be read. Wildcards are supported.
  - “Line delimiter” (optional): Specify the character separating records in the file;
  - “Log filtering” (optional): Add label filtering rules and filters to keep only the required records.

88Deckhouse

Search

System Management

Active alerts

Alert settings

Grafana Datasources

Custom Rules

Remote Writes

State

Availability metrics export

Managed Services

Postgres Classes

Trino Classes

Memcached Classes

Kafka Classes

Hive Metastore Classes

Valkey Classes

Delivery

Helm Addons

Helm Addon Charts

Helm Addon Repositories

Logging

Log Destinations

Logging Configs

Other

Other CRD

System / System Management / Logging / Logging Configs

Type: File

spec.type

Name\*

metadata.name

Storage

Log sending\*

spec.destinationRefs

Storages are defined in the "Log Delivery" section `ClusterLogDestination`, with which this log source will work. Fields with numeric and boolean types will be converted to strings.

+ ADD

File filter

File paths to read

spec.file.include

Wildcards are supported, for example `/var/log/*.log`

+ ADD

File paths that are not required to be read

spec.file.exclude

Wildcards are supported, for example `/var/log/*.log`

+ ADD

Delimiter between lines

spec.file.lineDelimiter

Example: `\n`

Log filtering

List of rules for filtering logs by their labels ⓘ

## Viewing API information

To view API information in Swagger format, click the question mark button in the lower-left corner of the interface in the drop-down user menu.

88Deckhouse

Search

System Management

Pinned sections

Overview

Modules

Deckhouse

Overview

Releases

Modules

Global settings

Platform Operator

Kubernetes

Projects

Projects

Project Templates

Namespaces

Nodes

Node Groups

Nodes

Node Group Configurations

V

?

+ YAML

System / System Management / Deckhouse

Updates

Releases

Settings

YAML

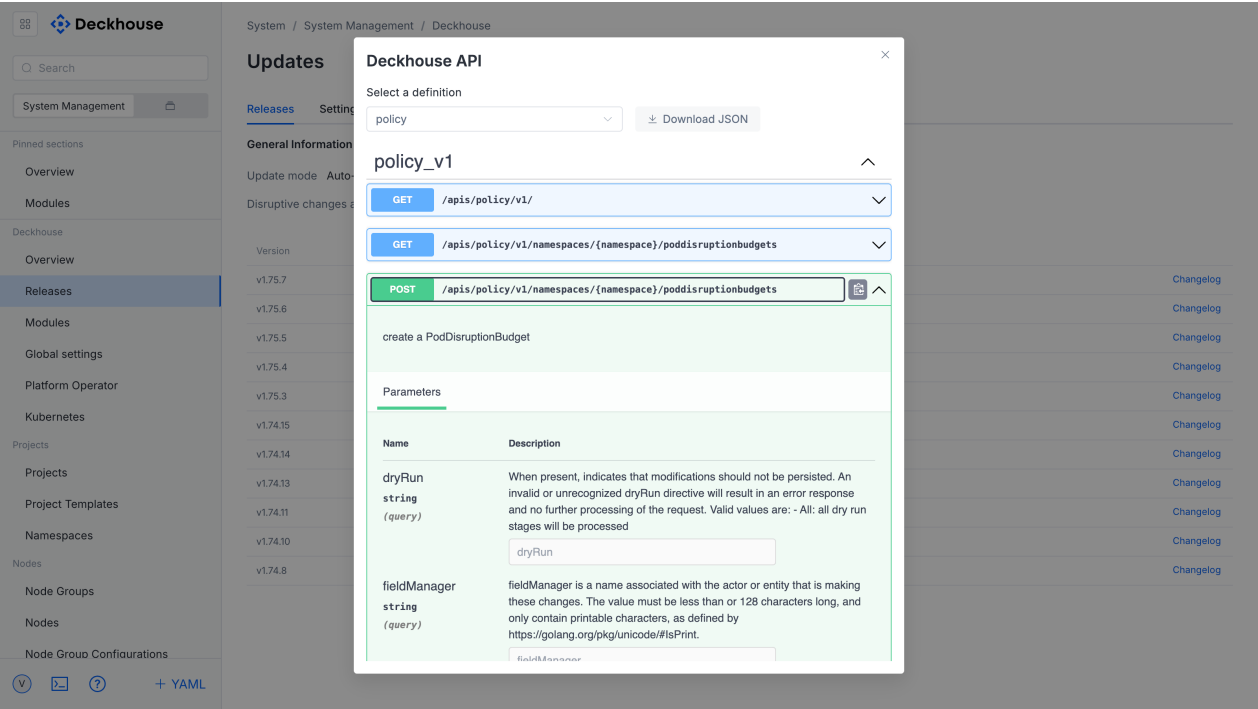
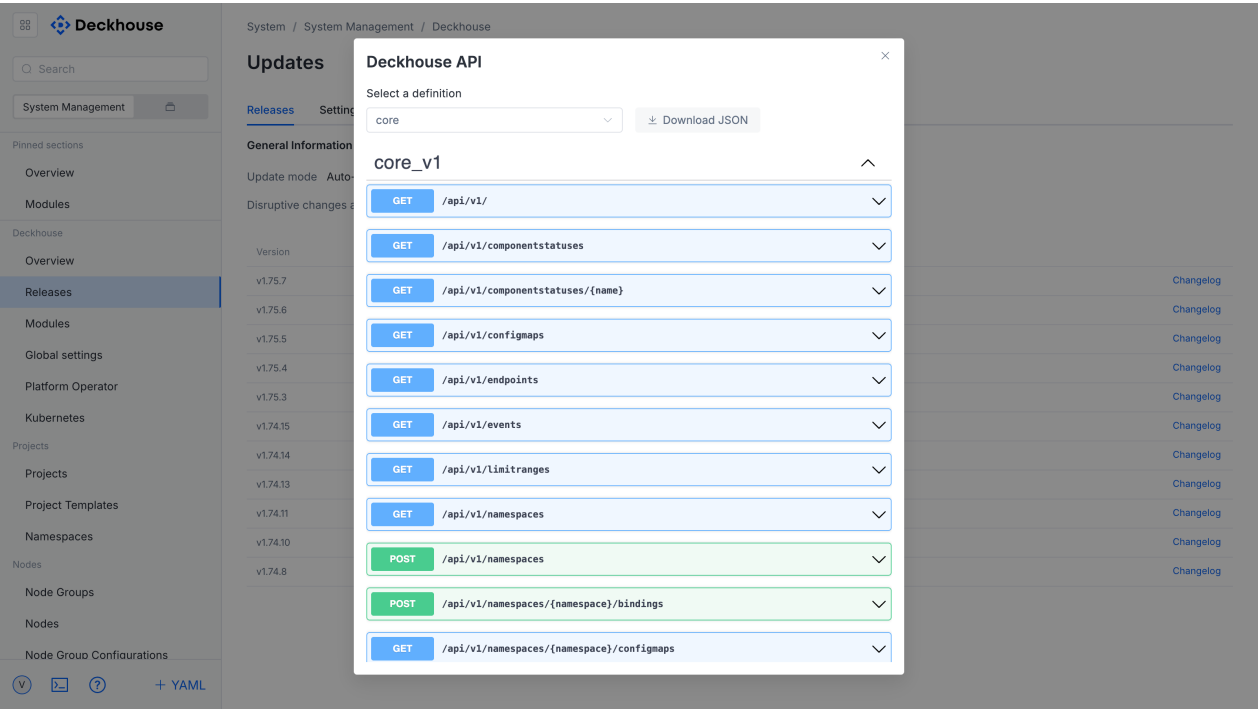
General Information

Update windows

Update mode `Auto-patch` ⓘ

Disruptive changes approval `Default`

| Version  | Status     | Released            | Installed           | Message                     |
|----------|------------|---------------------|---------------------|-----------------------------|
| v1.75.7  | DEPLOYED   | 15.05.2026 17:43:28 | 15.05.2026 18:43:39 | — <a href="#">Changelog</a> |
| v1.75.6  | SUPERSEDED | 08.05.2026 15:18:52 | 15.05.2026 18:43:39 | — <a href="#">Changelog</a> |
| v1.75.5  | SUPERSEDED | 29.04.2026 10:28:01 | 08.05.2026 16:49:06 | — <a href="#">Changelog</a> |
| v1.75.4  | SUPERSEDED | 21.04.2026 16:48:07 | 29.04.2026 12:58:09 | — <a href="#">Changelog</a> |
| v1.75.3  | SUPERSEDED | 09.04.2026 12:13:25 | 21.04.2026 19:18:13 | — <a href="#">Changelog</a> |
| v1.74.15 | SUPERSEDED | 30.03.2026 14:02:56 | 09.04.2026 12:13:27 | — <a href="#">Changelog</a> |
| v1.74.14 | SUPERSEDED | 23.03.2026 14:19:32 | 30.03.2026 14:02:56 | — <a href="#">Changelog</a> |
| v1.74.13 | SUPERSEDED | 13.03.2026 11:38:15 | 23.03.2026 15:49:35 | — <a href="#">Changelog</a> |
| v1.74.11 | SUPERSEDED | 06.03.2026 13:44:18 | 13.03.2026 11:38:15 | — <a href="#">Changelog</a> |
| v1.74.10 | SUPERSEDED | 02.03.2026 17:23:38 | 06.03.2026 13:44:19 | — <a href="#">Changelog</a> |
| v1.74.8  | SUPERSEDED | 20.02.2026 12:43:21 | 02.03.2026 17:53:39 | — <a href="#">Changelog</a> |



Additional resources

- Global parameters
- Module prometheus documentation

Grafana web interface

Monitoring system web UI

Grafana is used as the web UI for the monitoring system.

Main page

The main page of Grafana displays key information about the cluster and its main components.

In the top left corner, you'll find details about the main cluster components: the containerd version, the Linux distribution the cluster is running on, and the versions of Grafana, Prometheus, and other components.

The top right corner features convenient graphical indicators for key metrics including the number of

nodes in the cluster, the number of running Pods, and other cluster entities.

For more detailed information, hover the mouse cursor over any element to display a tooltip.

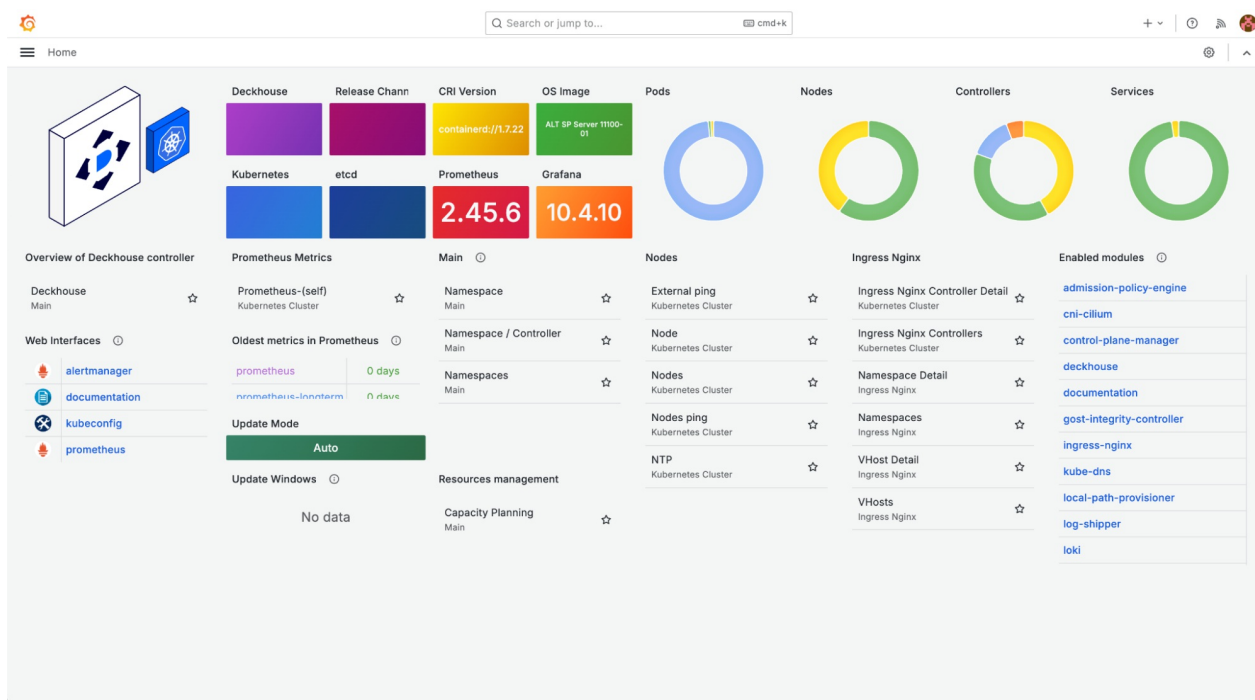
Below the component info blocks, there are quick links to dashboards for specific cluster components, such as node monitoring, system resource usage, and network interaction statistics.

On the left side, you'll find a section with links to the cluster's available web interfaces, as well as a block showing the cluster update method and the scheduled update windows (if configured).

In the top left corner of the main page is a button that opens the side menu, which contains links to Grafana's core sections.

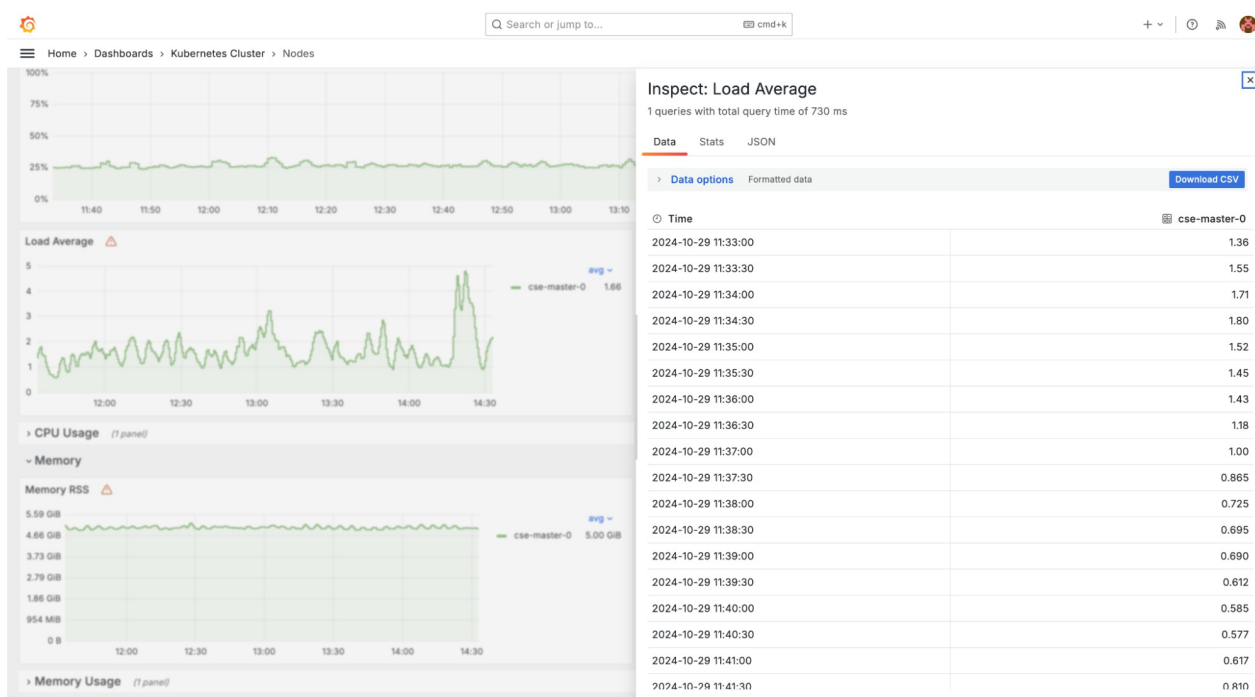
Clicking the **Dashboards** tab opens a list of all available Deckhouse dashboards, grouped by category.

These dashboards have a nested structure and are grouped by purpose: cluster applications (**Applications**), network interaction (**Ingress Nginx**), cluster parameters (**Kubernetes Cluster**), **Main**, and **Security**.



## Working with dashboards

A dashboard is a page with tables and graphs that display information about a selected cluster component.





## Data filtering

At the top of the page, below the title and the quick link to the parent category, you can find a filter block. This block lets you customize the displayed data by excluding irrelevant information or focusing on a specific component.



Q Search or jump to...

Home > Dashboards > Kubernetes Cluster > Nodes ☆

Prometheusdefault ▾NodeAll ▾

Average

| Node name                    | Cores | Load Average | CPU Usage |
|------------------------------|-------|--------------|-----------|
| <a href="#">cse-worker-0</a> | 3     | 1.21         | 13.60%    |
| <a href="#">cse-master-0</a> | 5     | 1.94         | 25.62%    |
| <a href="#">cse-master-1</a> | 4     | 1.06         | 14.48%    |
| <a href="#">cse-master-2</a> | 4     | 1.22         | 22.36%    |
| <a href="#">cse-worker-1</a> | 3     | 0.30         | 7.89%     |

For example, in the dashboard showing cluster nodes, you can use the filter to display only one of the three nodes, excluding the others.



Q Search or jump to...

Home > Dashboards > Kubernetes Cluster > Nodes ☆

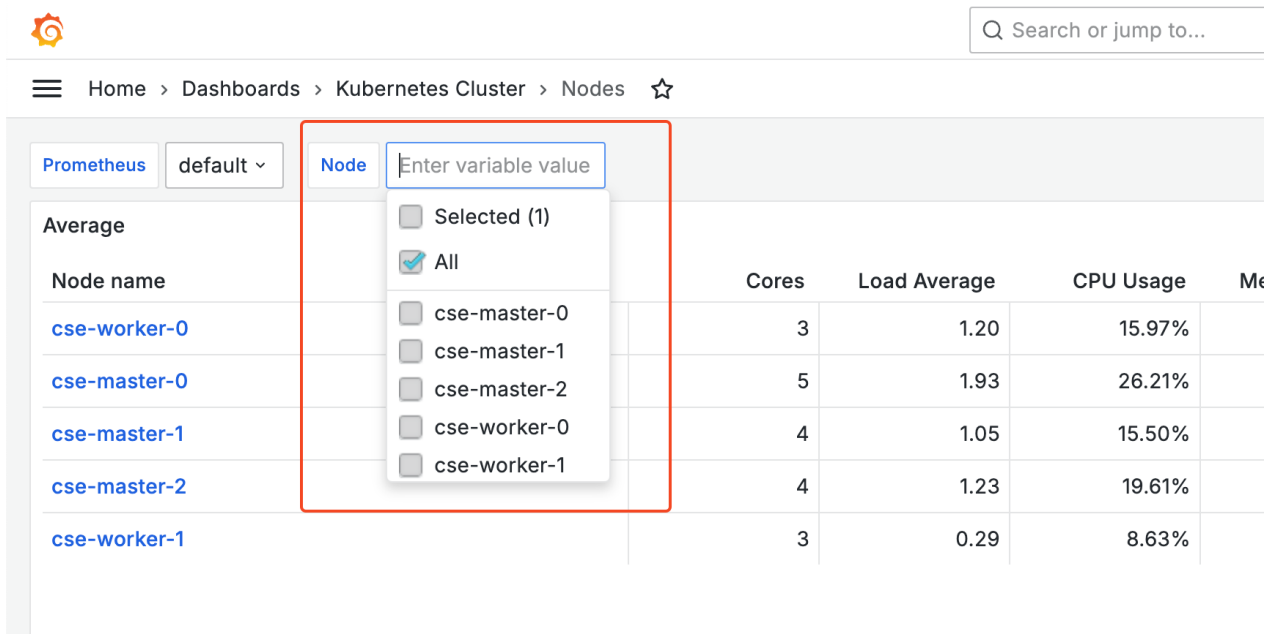
Prometheusdefault ▾Node

Average

☐ Selected (1)  
☒ All  
☐ cse-master-0  
☐ cse-master-1  
☐ cse-master-2  
☐ cse-worker-0  
☐ cse-worker-1

| Node name                    | Cores | Load Average | CPU Usage | Mem |
|------------------------------|-------|--------------|-----------|-----|
| <a href="#">cse-worker-0</a> | 3     | 1.20         | 15.97%    |     |
| <a href="#">cse-master-0</a> | 5     | 1.93         | 26.21%    |     |
| <a href="#">cse-master-1</a> | 4     | 1.05         | 15.50%    |     |
| <a href="#">cse-master-2</a> | 4     | 1.23         | 19.61%    |     |
| <a href="#">cse-worker-1</a> | 3     | 0.29         | 8.63%     |     |

Once a filter is selected, the dashboard will update immediately to show data that matches the chosen filter parameters.

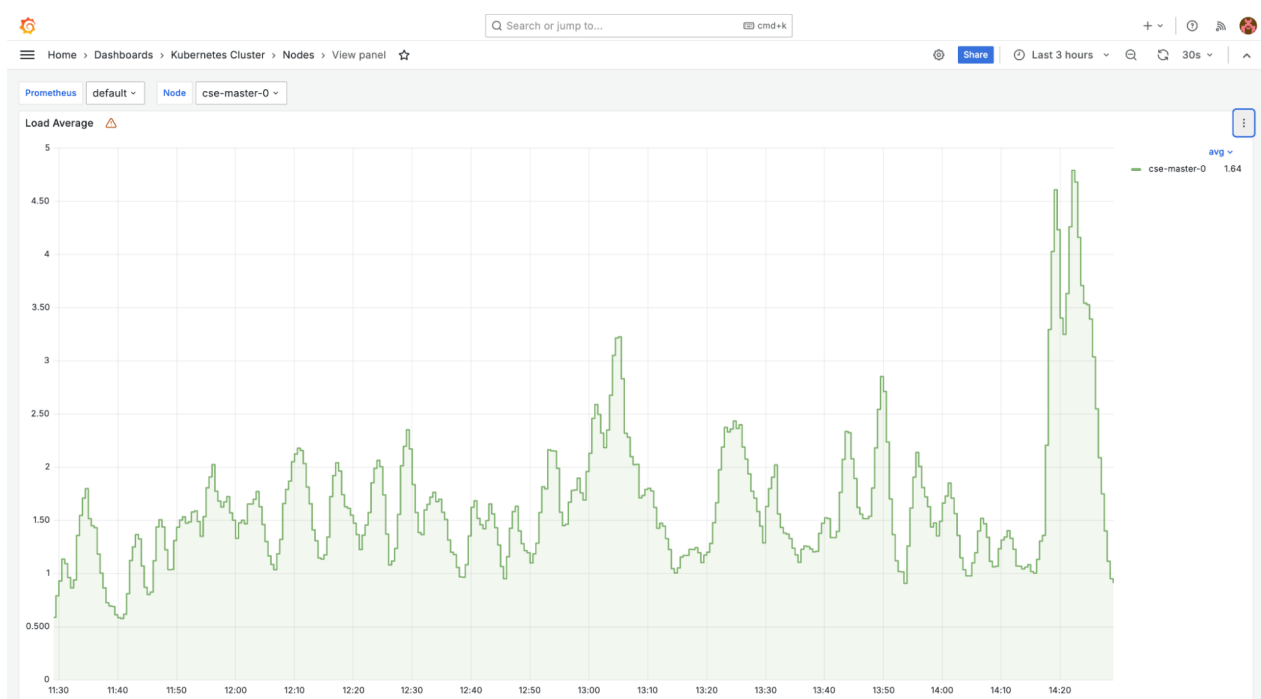


## Working with data

Each chart on a dashboard can be viewed in more detail. To do this, hover the mouse cursor over the top right corner of the chart block, click the three-dot menu button, and select **View**.



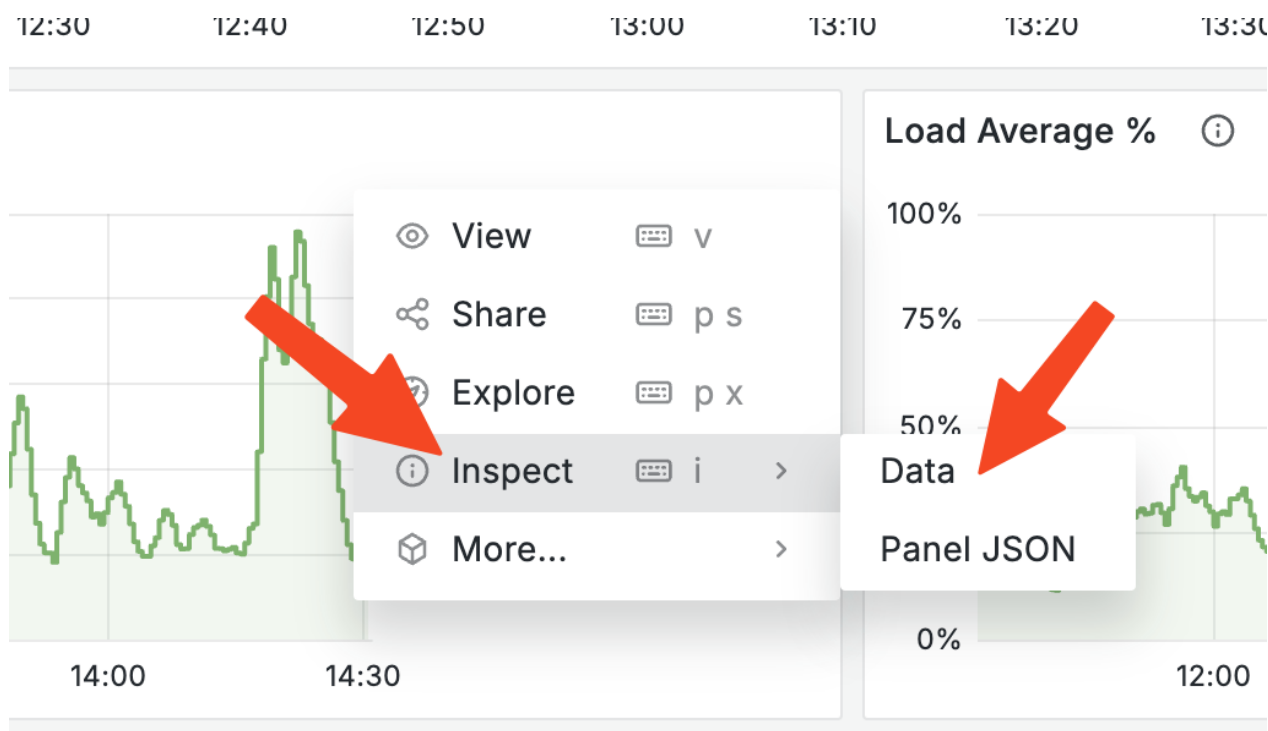
The selected chart will expand to full-screen view.



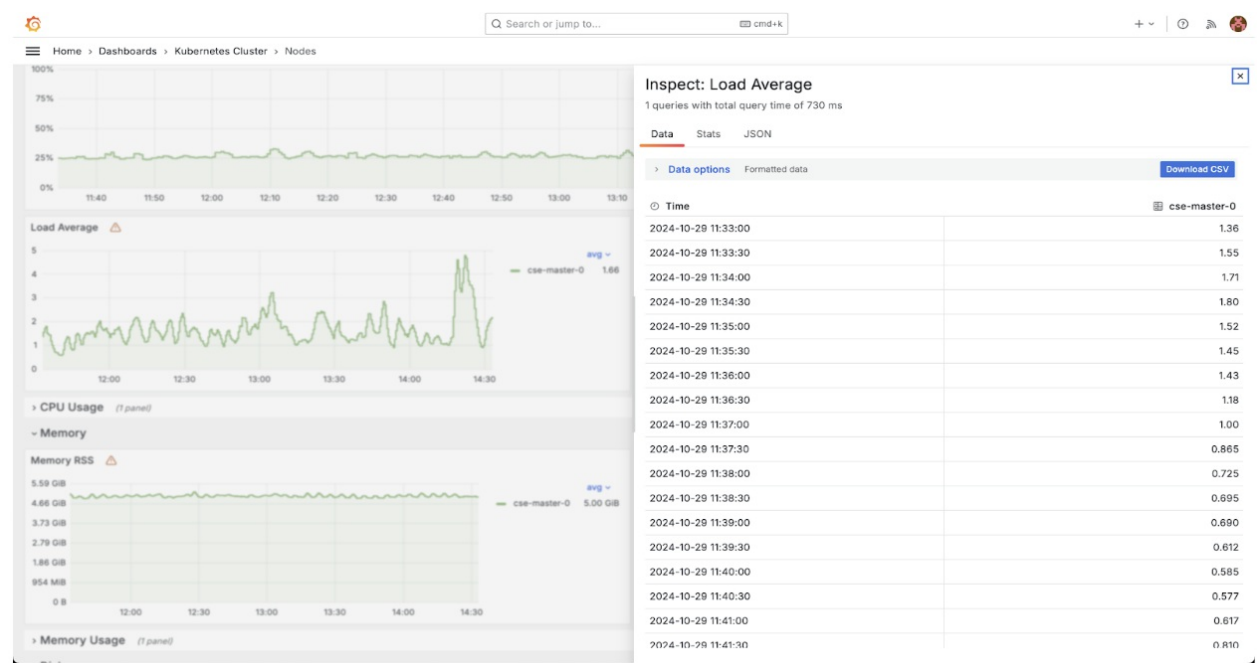
Here you can explore detailed data for a specific point in time. Hover over the graph. A red horizontal line will appear, and a tooltip will display the timestamp and value at that moment:



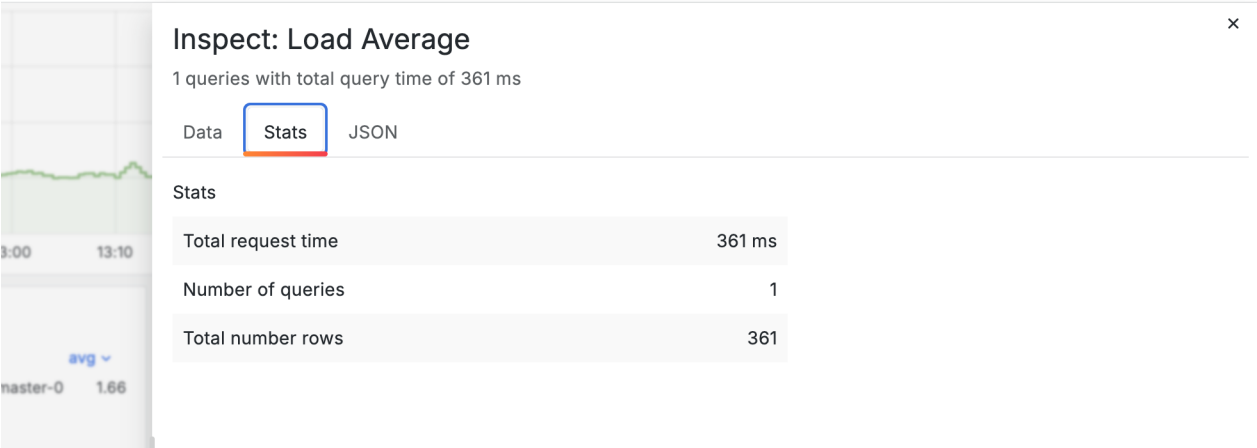
To return to the previous page, press **Esc**. To view the raw data behind the graph, click the three-dot menu in the top-right corner of the chart block and select **Inspect**, then choose the **Data** tab that corresponds to the query.



A panel with detailed record content will open on the right side of the page.

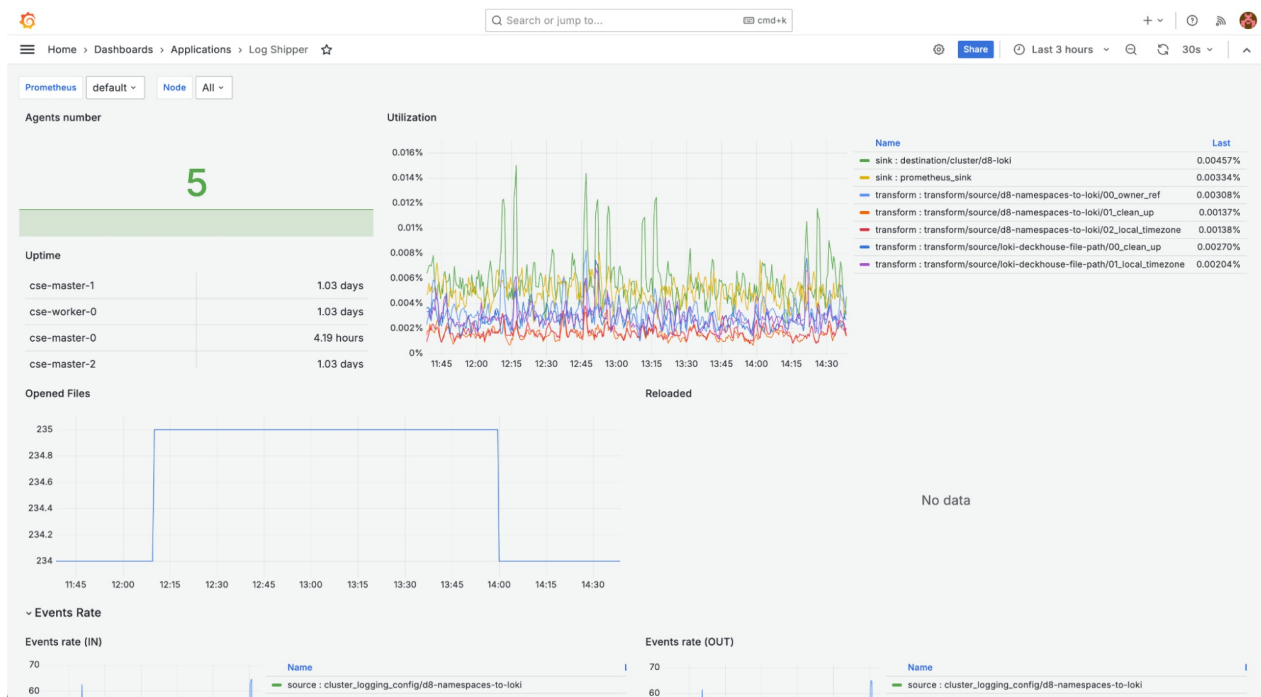


The panel displays all the data used to build the graph. You can also download it as a CSV file and view basic statistics (such as the total number of records) by switching to the **Stats** tab in the data panel.

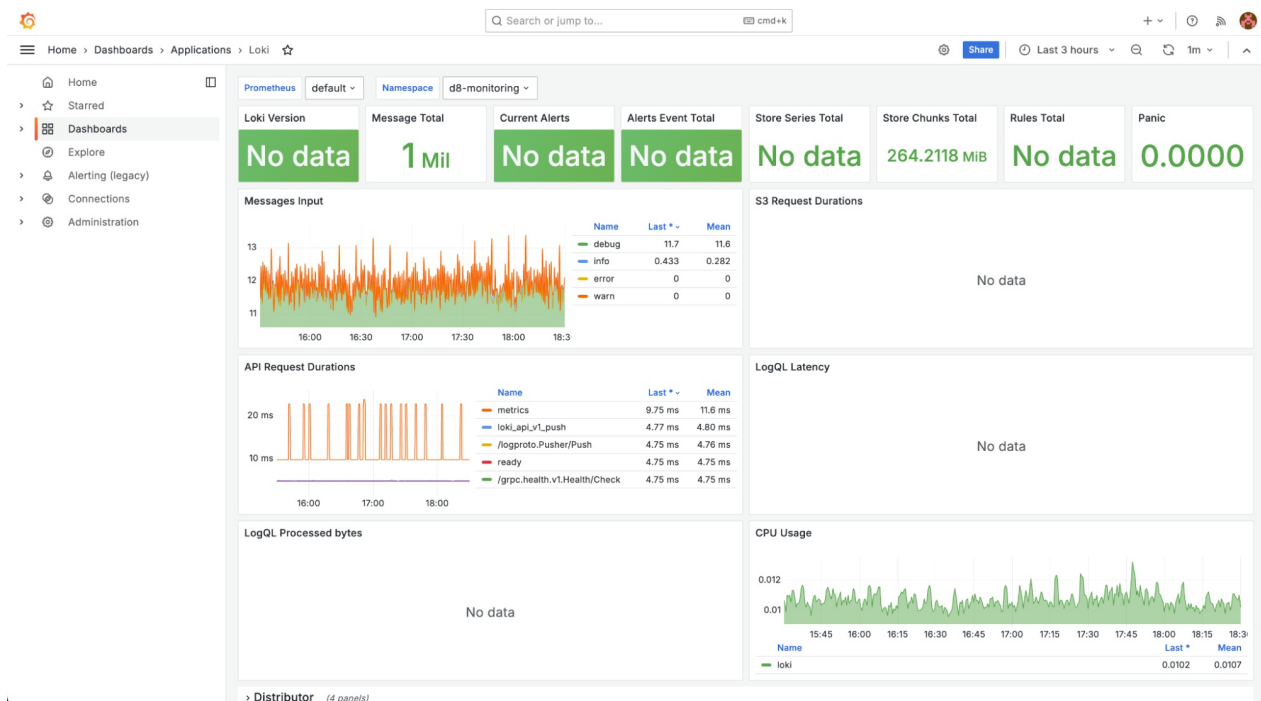


## Dashboard descriptions

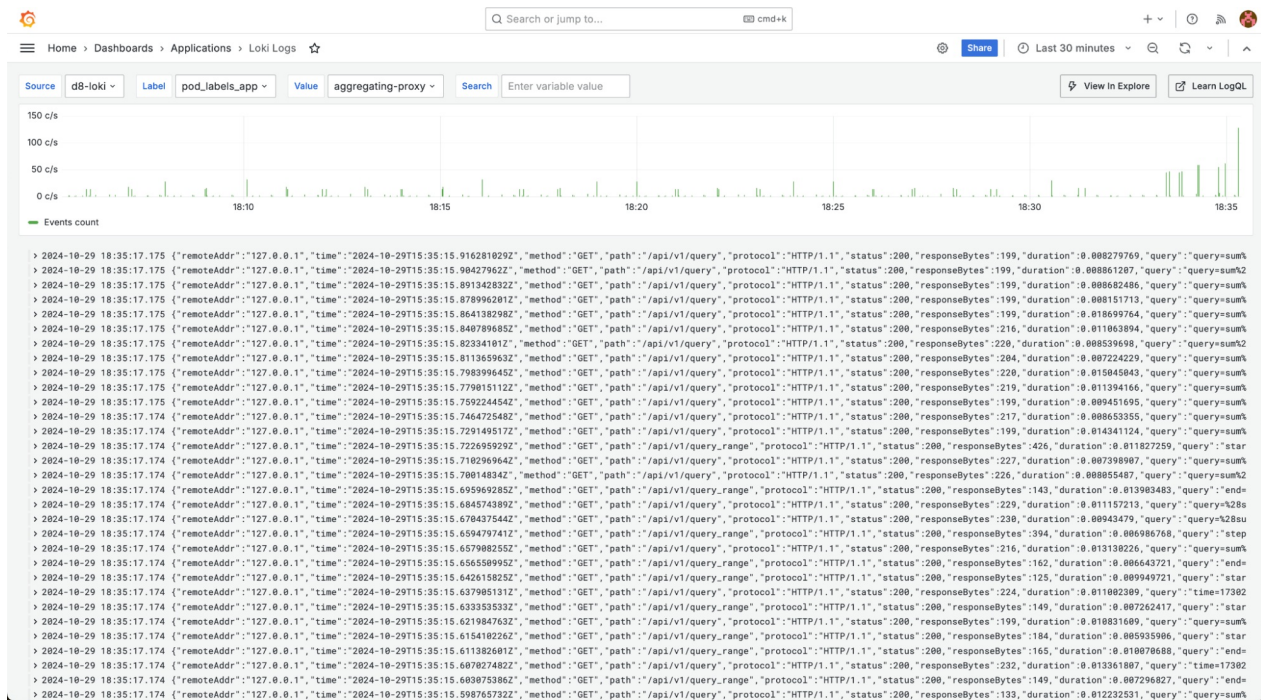
“Applications – Log Shipper” dashboard shows the status of the log-shipper module, including the number of agents running on nodes and their load.



“Applications – Loki” dashboard shows the status of the loki module.

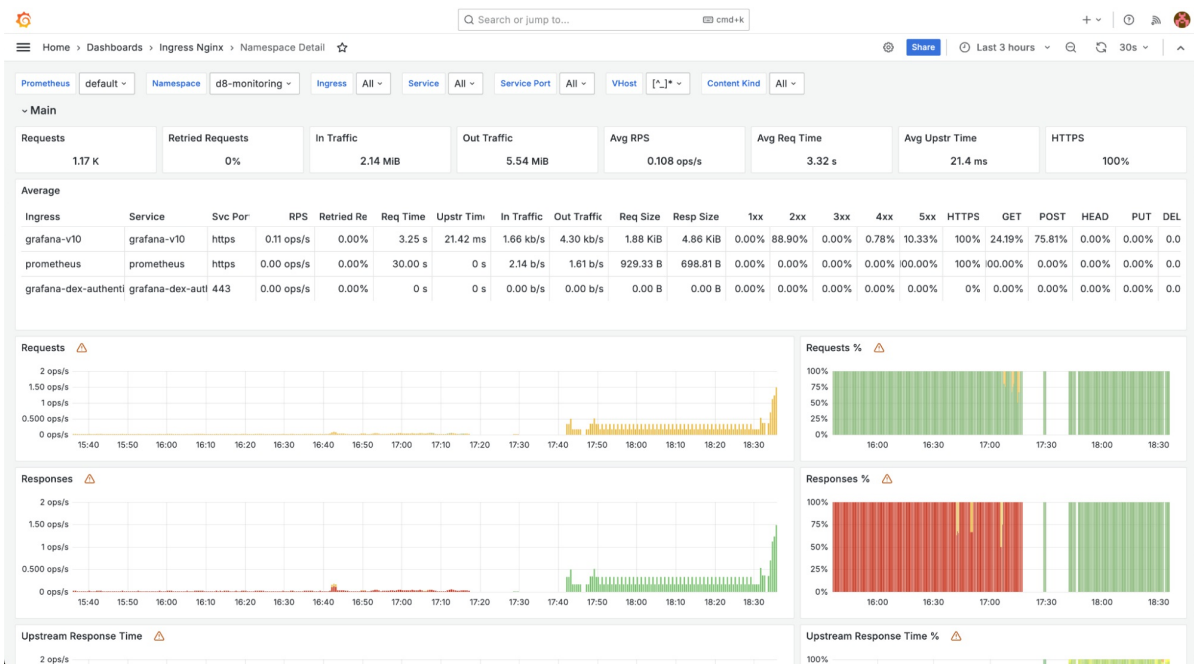


“Applications – Loki Logs” dashboard displays logs from the loki module.

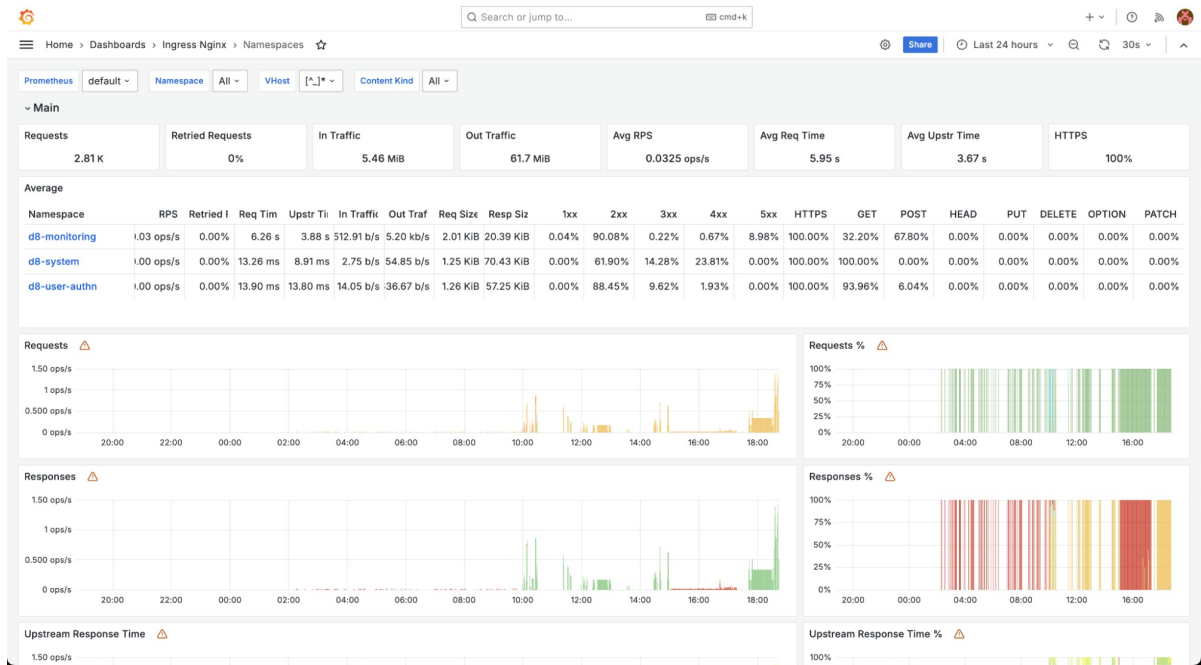


The Ingress Nginx group contains dashboards related to Ingress controllers:

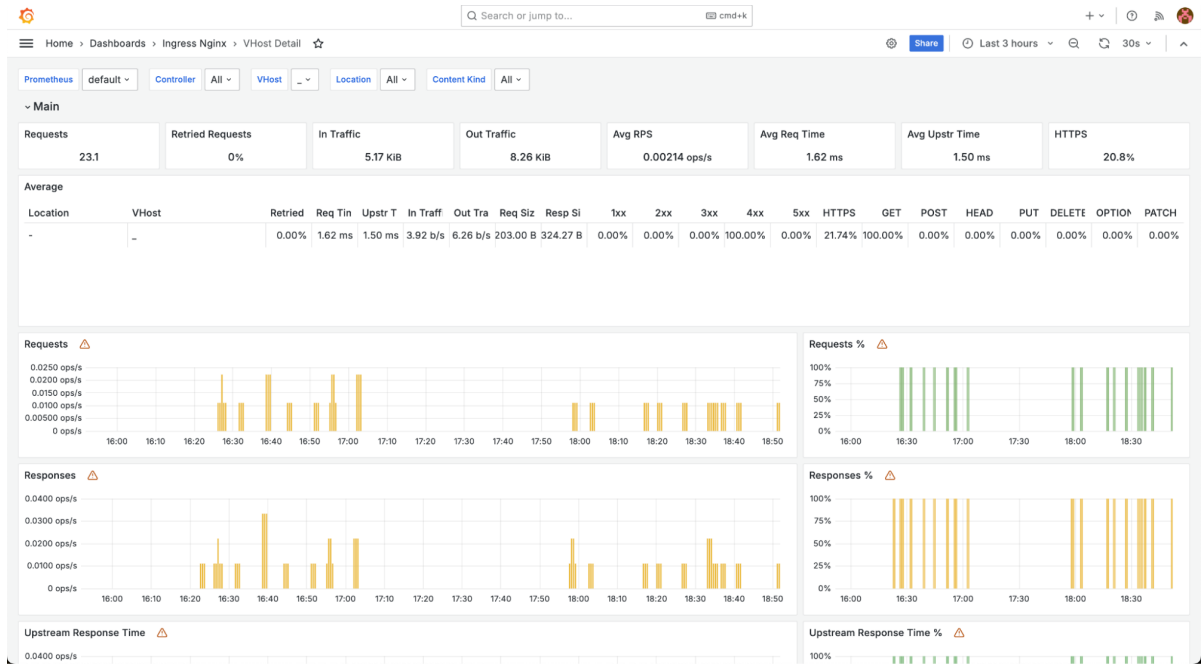
- “Namespace Detail” displays a breakdown of components within a specific namespace. You can filter by namespace, Ingress, Service, and other parameters.



- “Namespaces” shows Ingress controller data grouped by cluster namespaces. Filters allow you to select a specific namespace, virtual hosts, and content types.

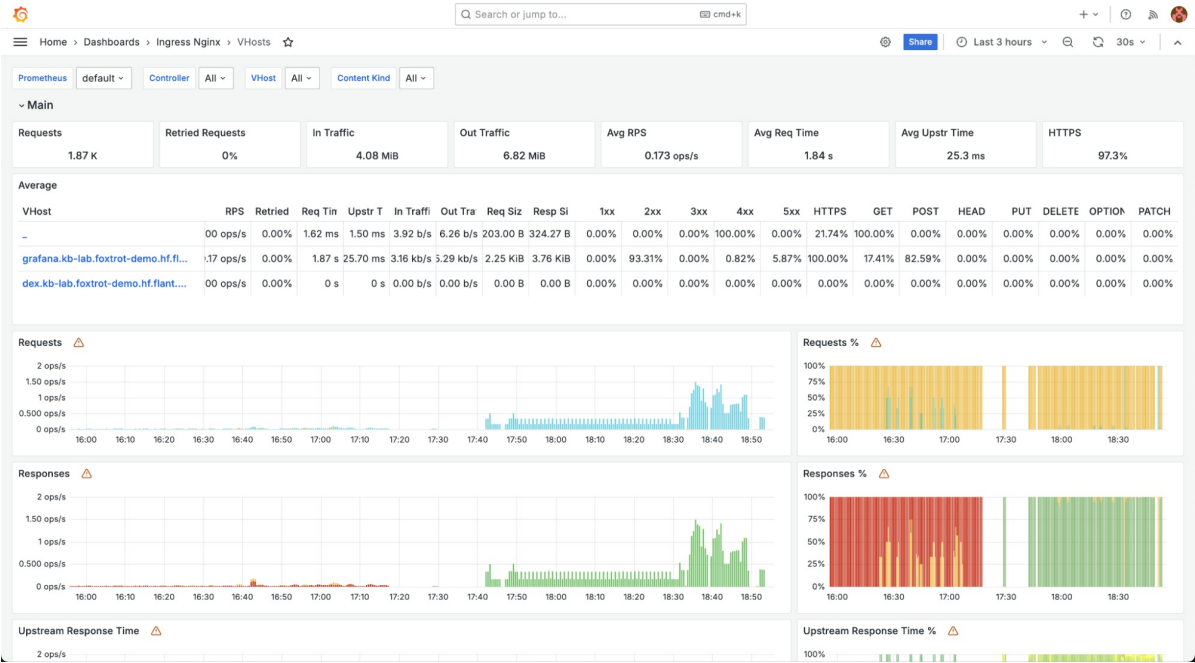


- “VHost Detail” shows detailed information on the Ingress controller by virtual hosts.



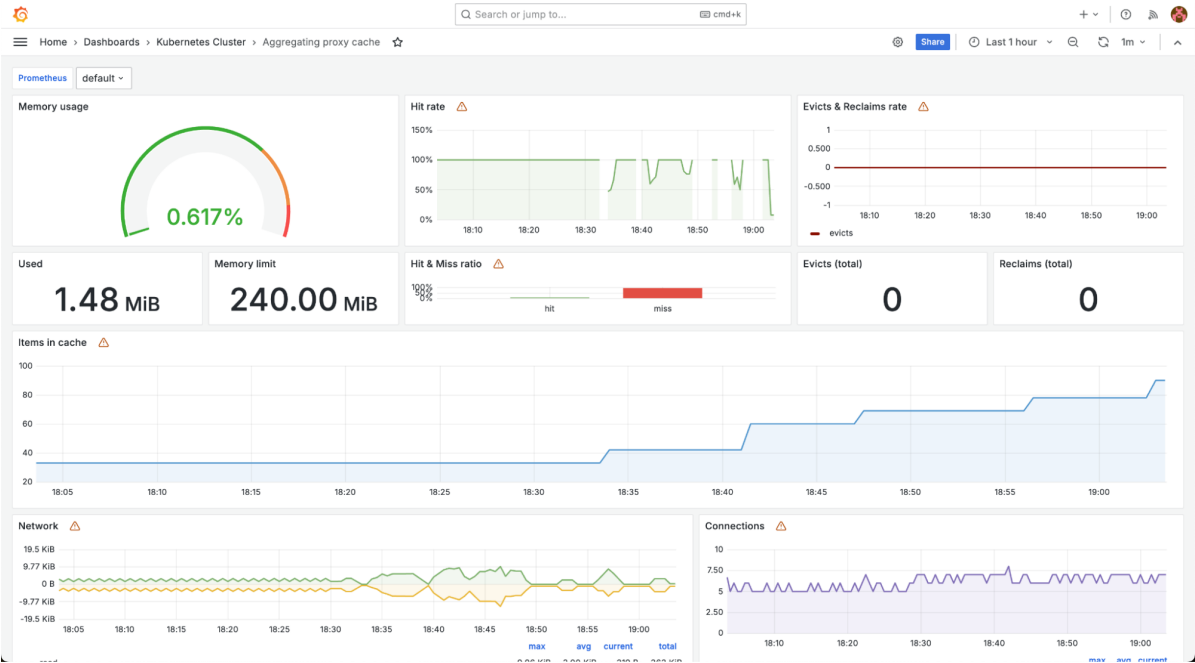
- “VHost” displays summary data on the Ingress controller by virtual hosts. You can filter by a specific virtual host.





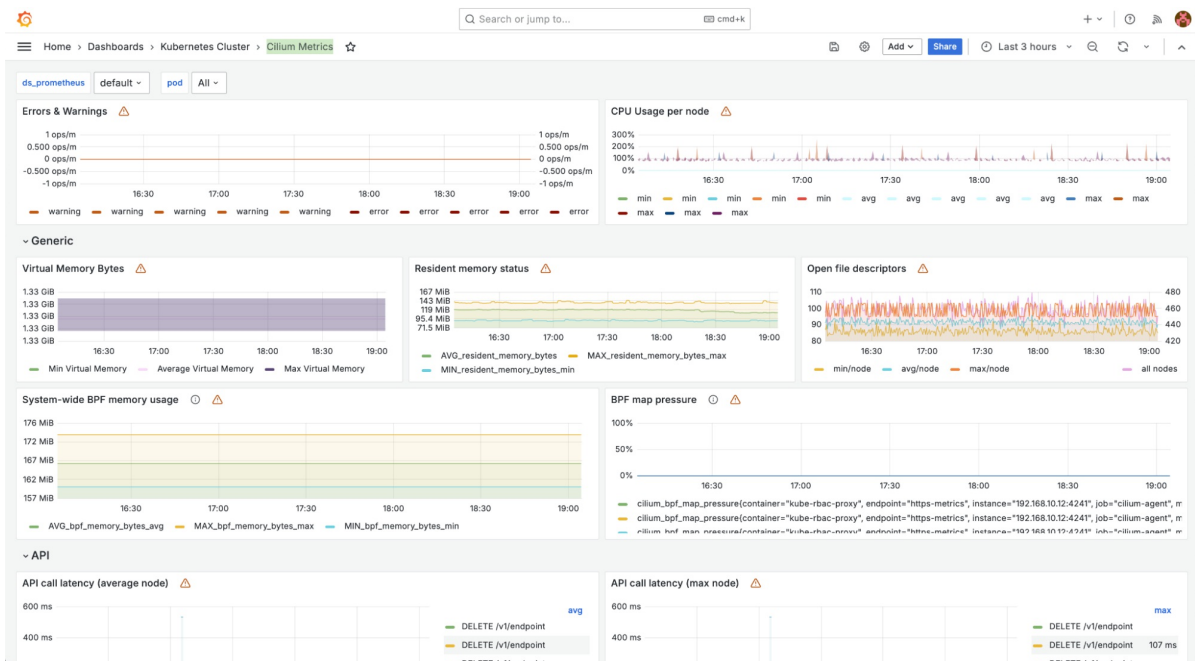
The Kubernetes Cluster group contains dashboards related to the Kubernetes cluster:

- “Aggregating Proxy Cache” displays a summary of the resources consumed by the proxy server.

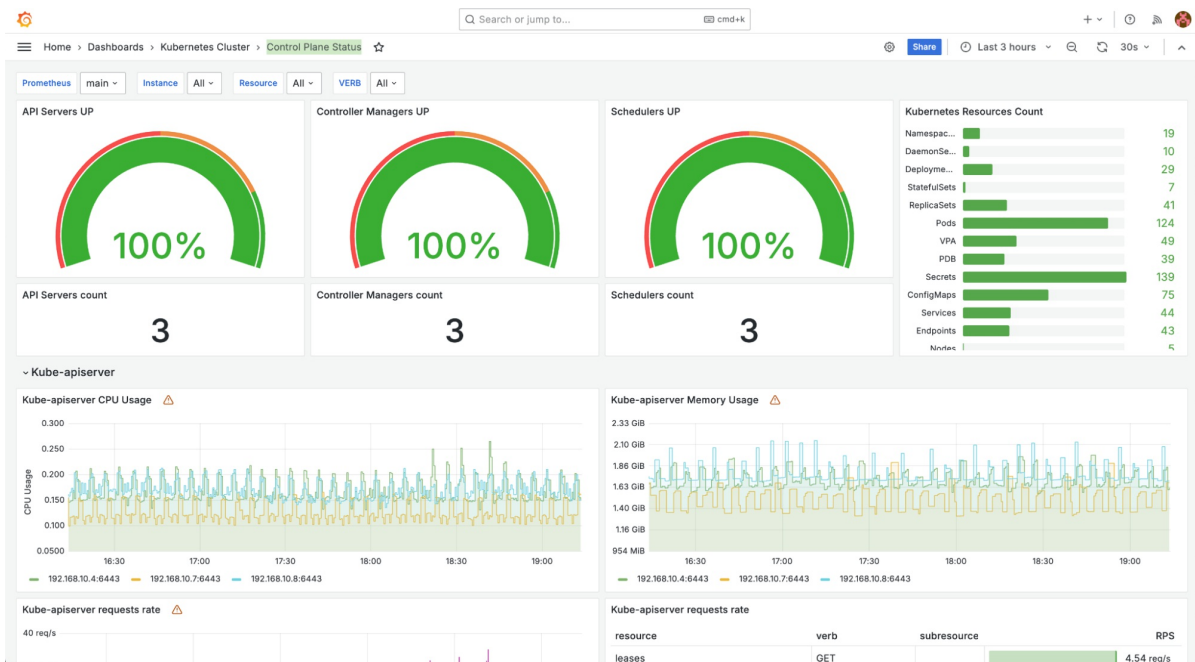


- “Cilium Metrics” displays metrics from the cni-cilium module.

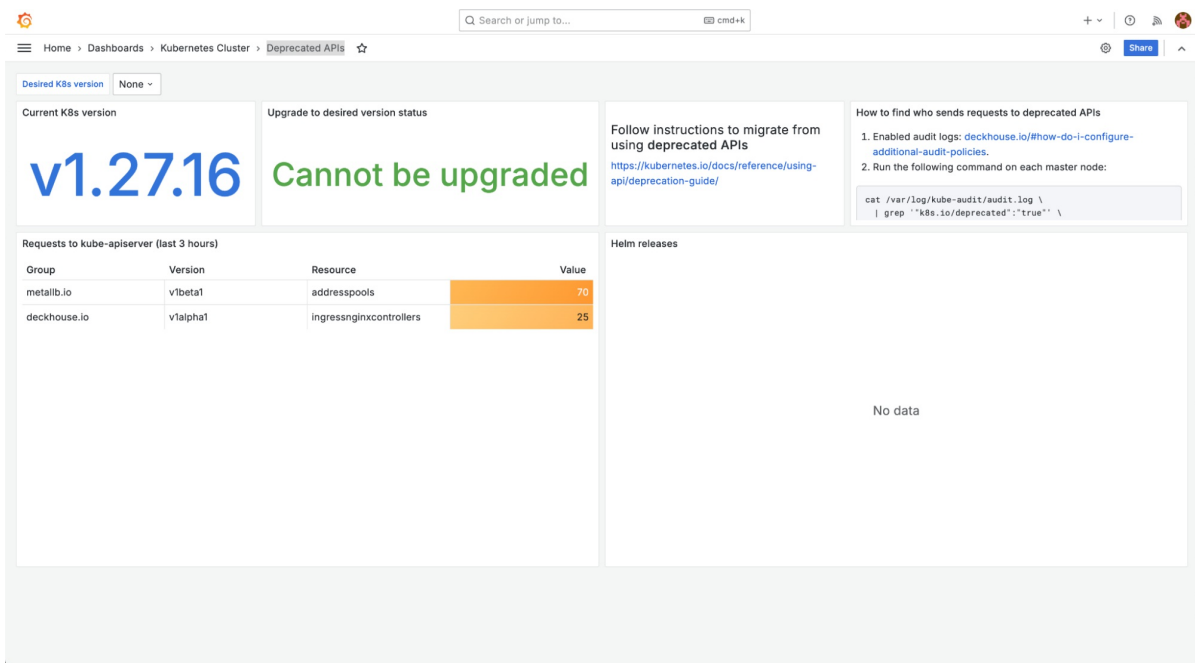




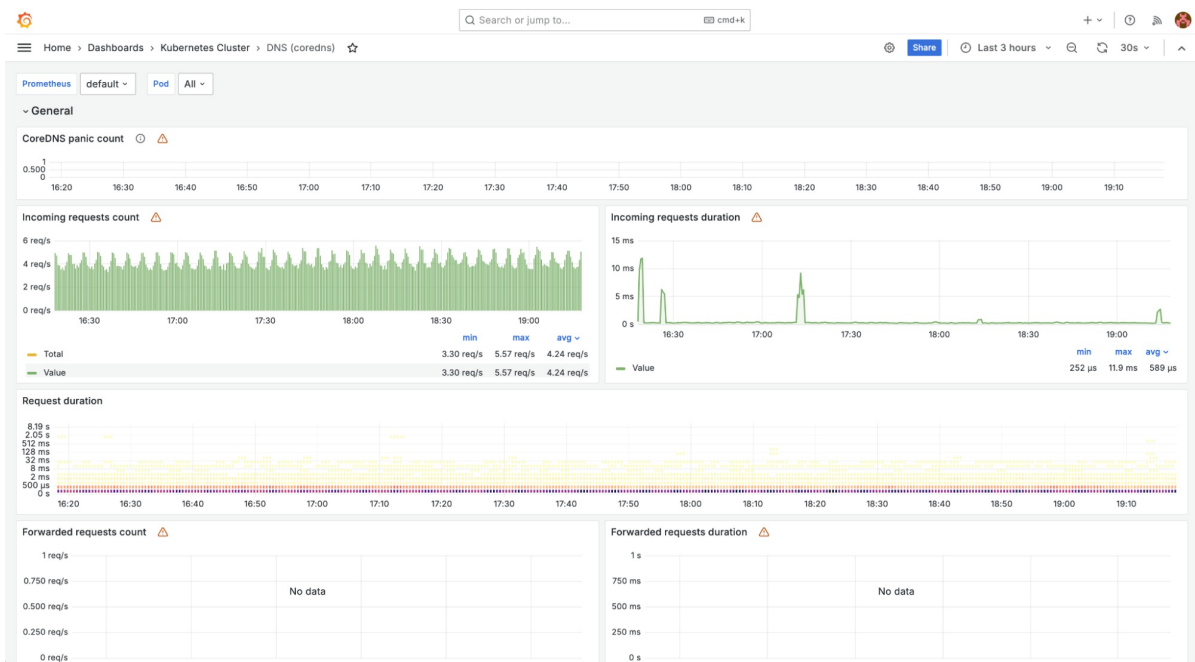
- “Control Plane Status” shows the status of the cluster control plane.



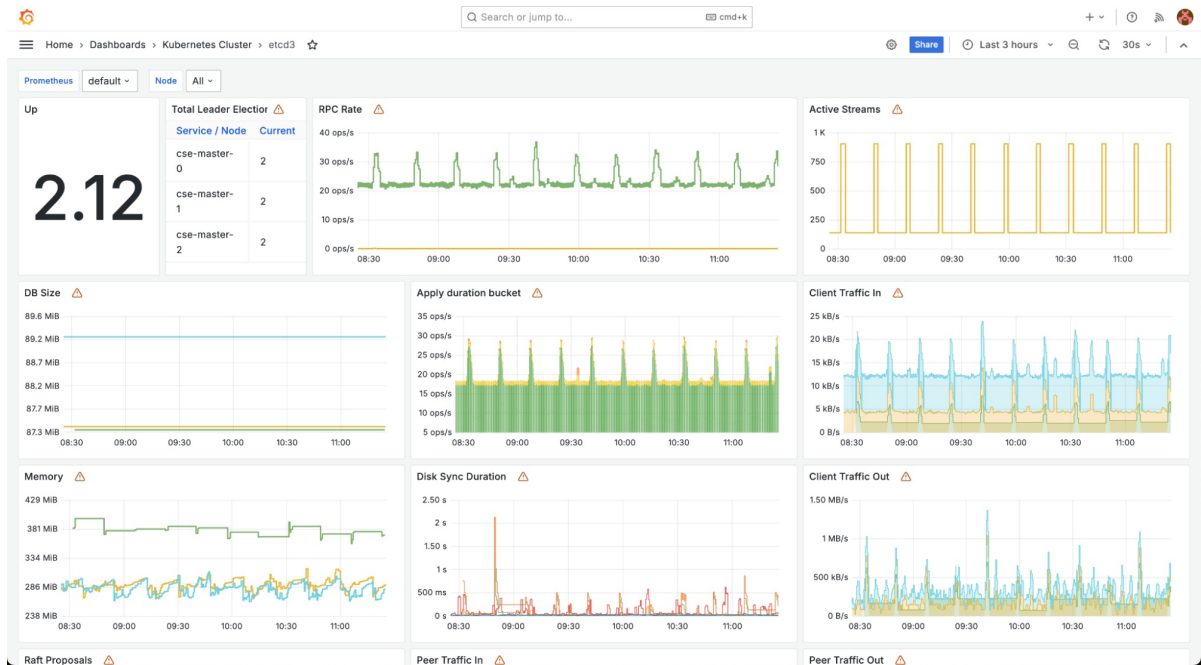
- “Deprecated APIs” displays the state of Kubernetes APIs that are currently deprecated. The dashboard also includes migration instructions and requests made to these API endpoints.



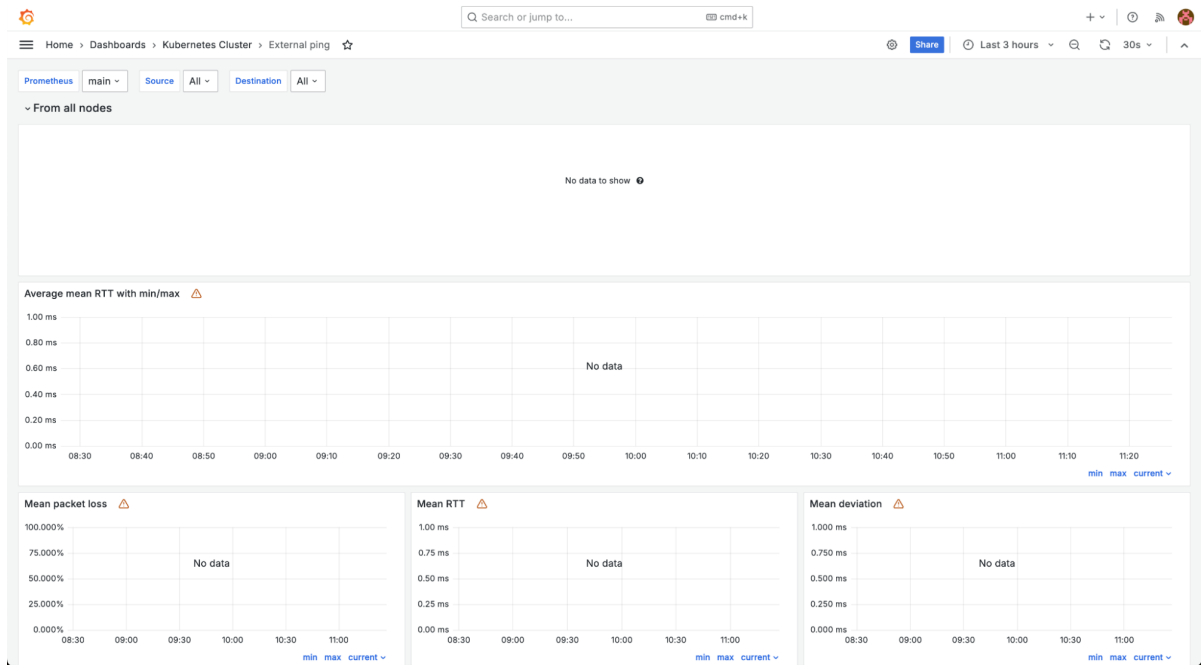
- “DNS (coredns)” dashboard displays data on the operation of the CoreDNS component.



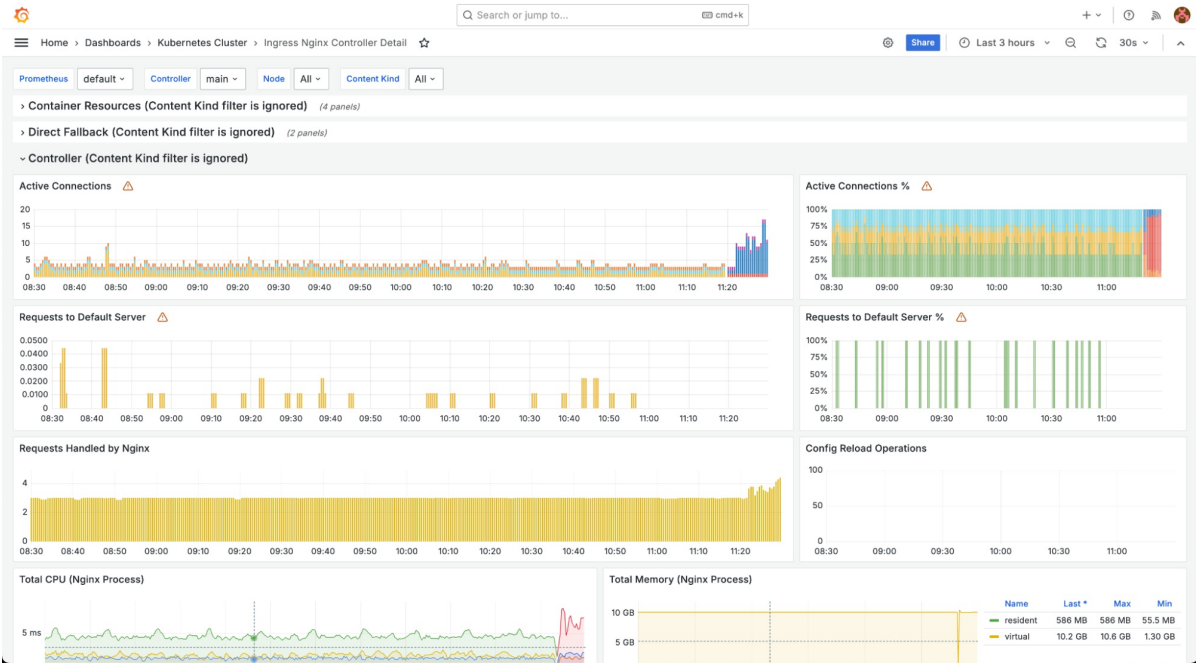
- “etcd3” dashboard shows the status of the etcd database.



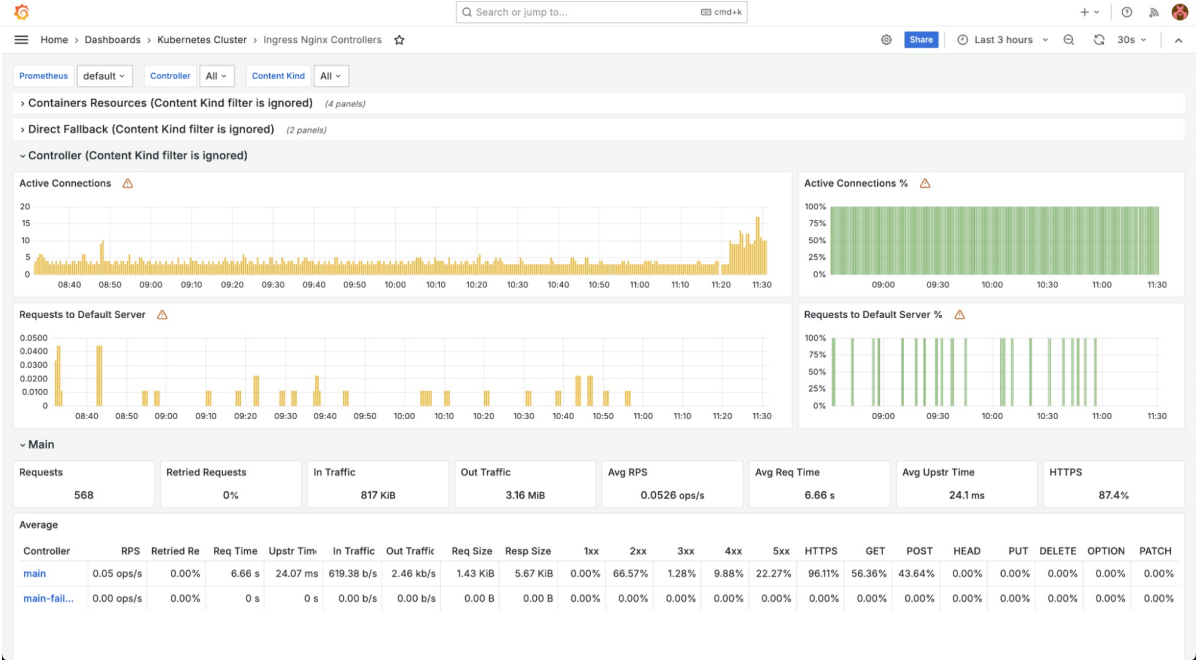
- “External ping” dashboard provides statistics on external requests.



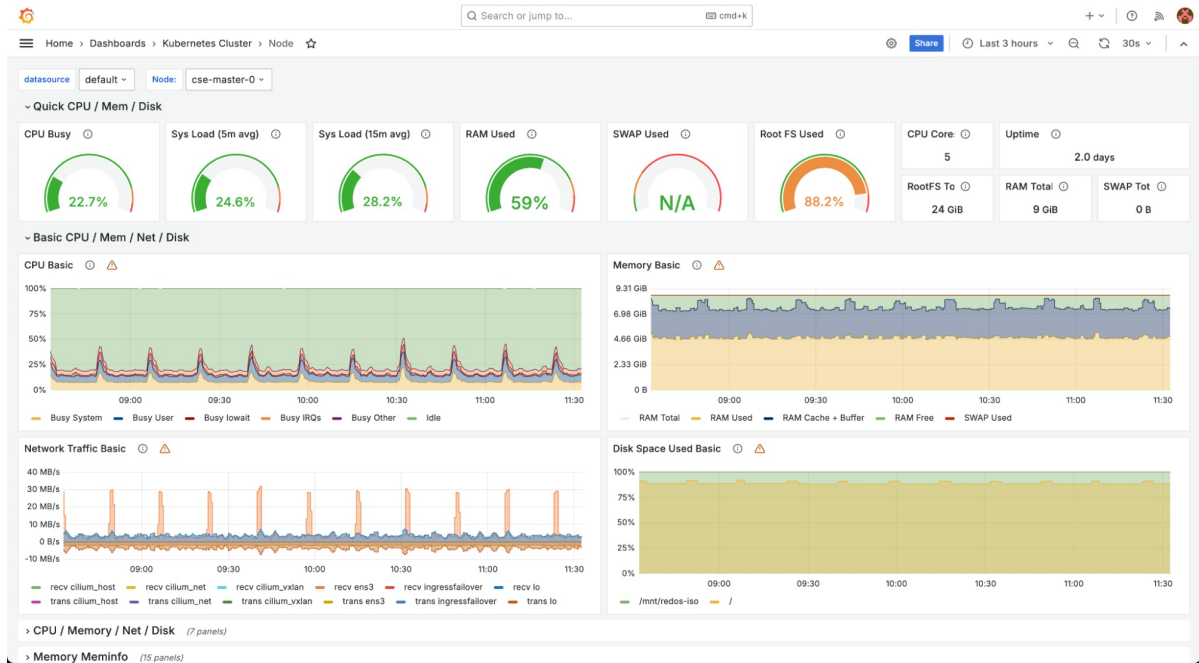
- “Ingress Nginx Controller Detail” dashboard displays parameters of the Ingress controller.



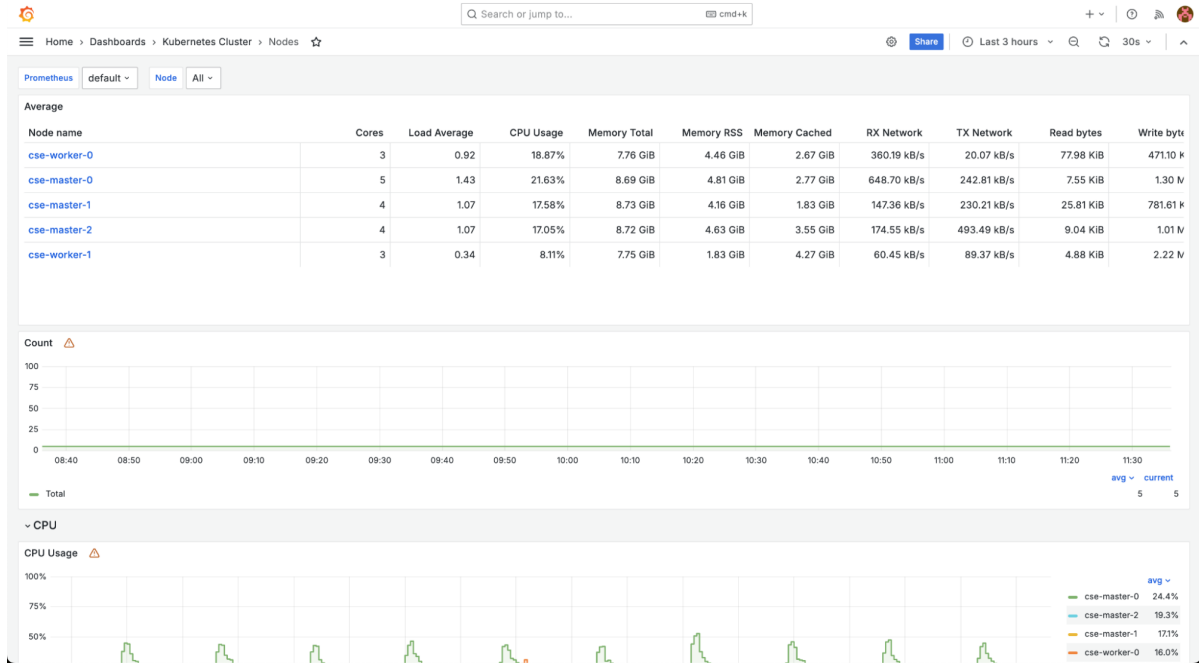
- “Ingress Nginx Controllers” dashboard shows detailed data on the cluster’s Ingress controllers.



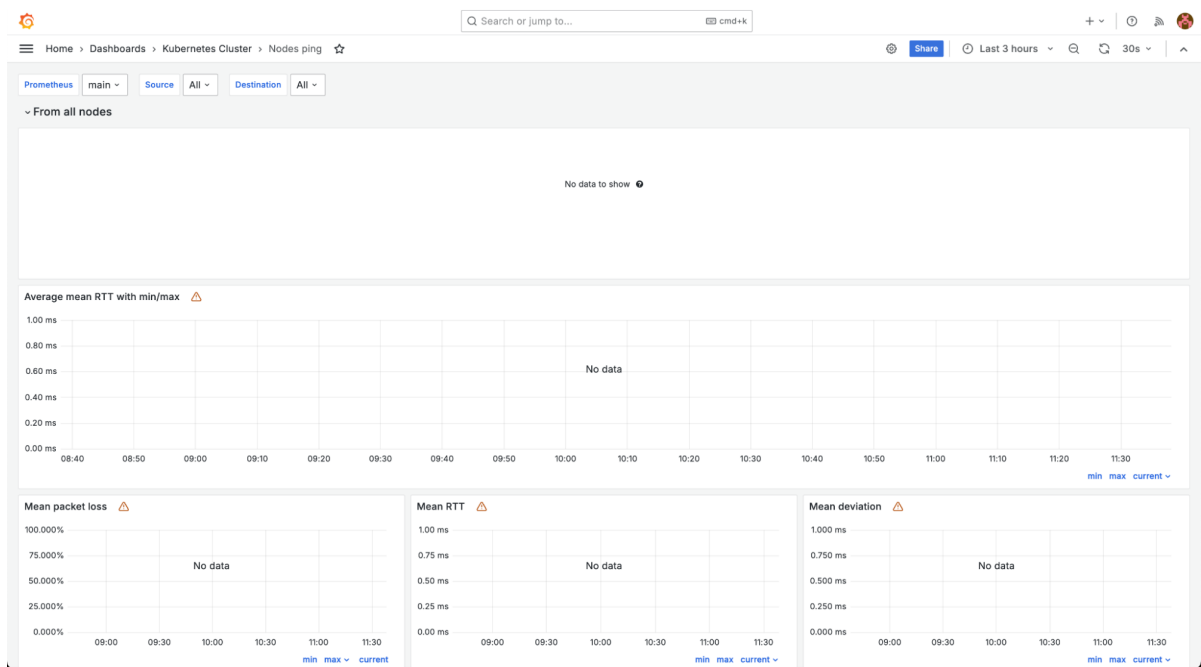
- “Node” dashboard displays information about individual node performance. You can select a specific node in the filters.



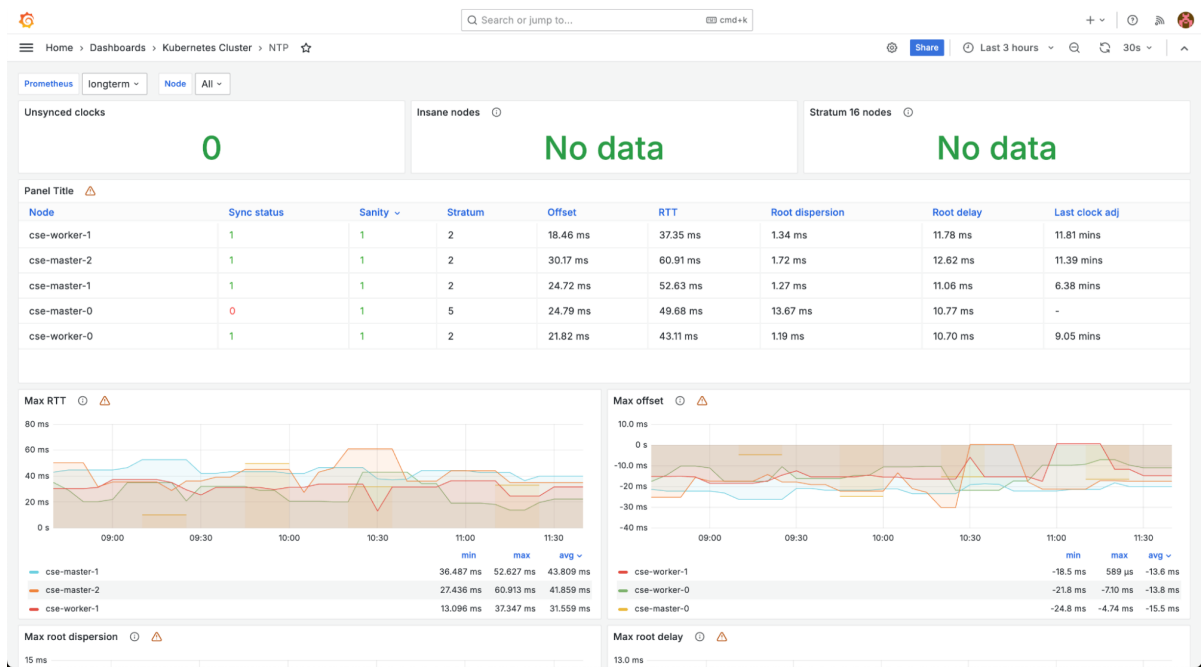
- “Nodes” dashboard shows aggregated data on all cluster nodes. The filters let you select a specific node.



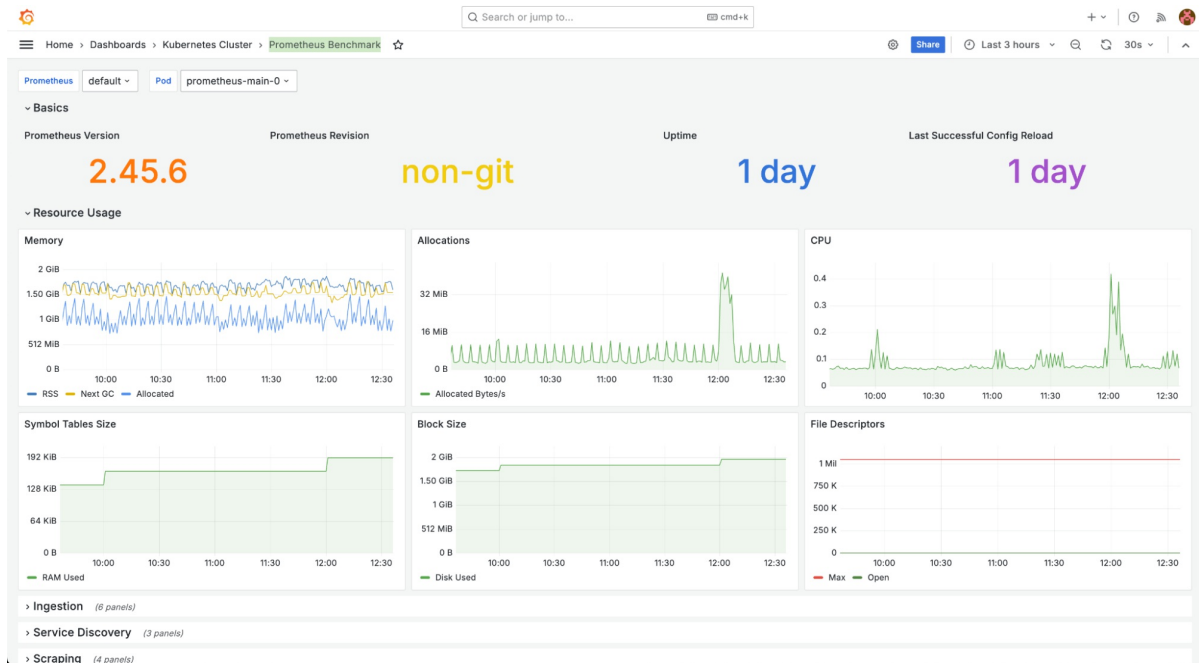
- “Nodes ping” dashboard displays ping results for cluster nodes.



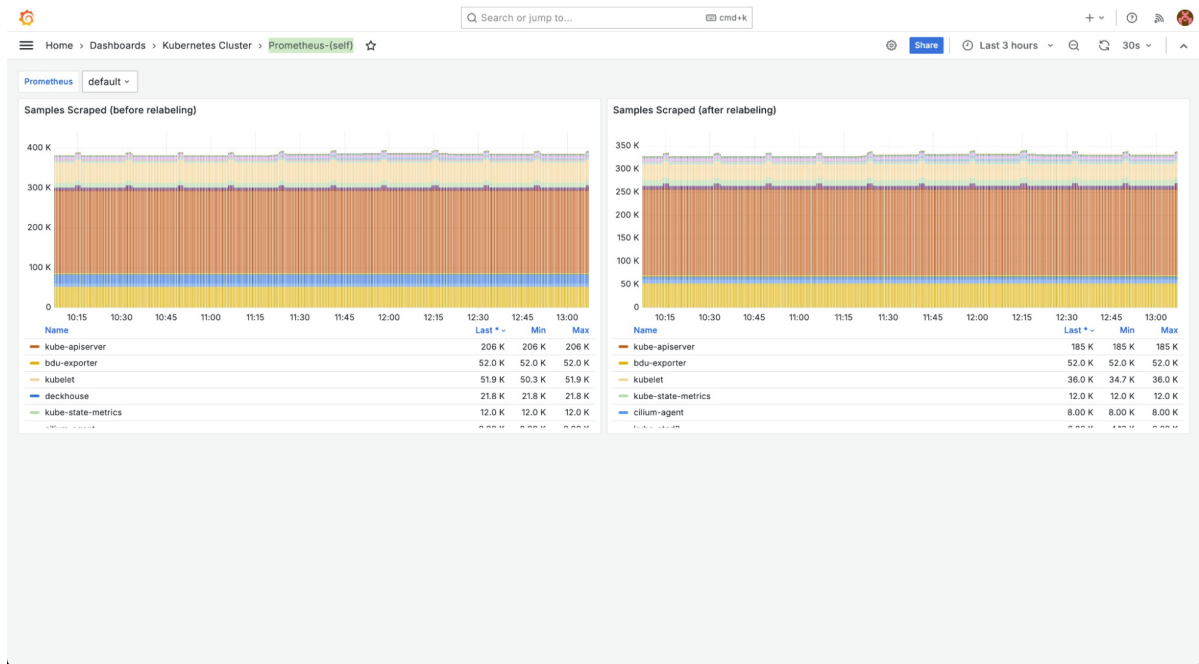
- “NTP” dashboard shows a status of the time synchronization server.



- “Prometheus Benchmark” dashboard shows a status of the prometheus module.



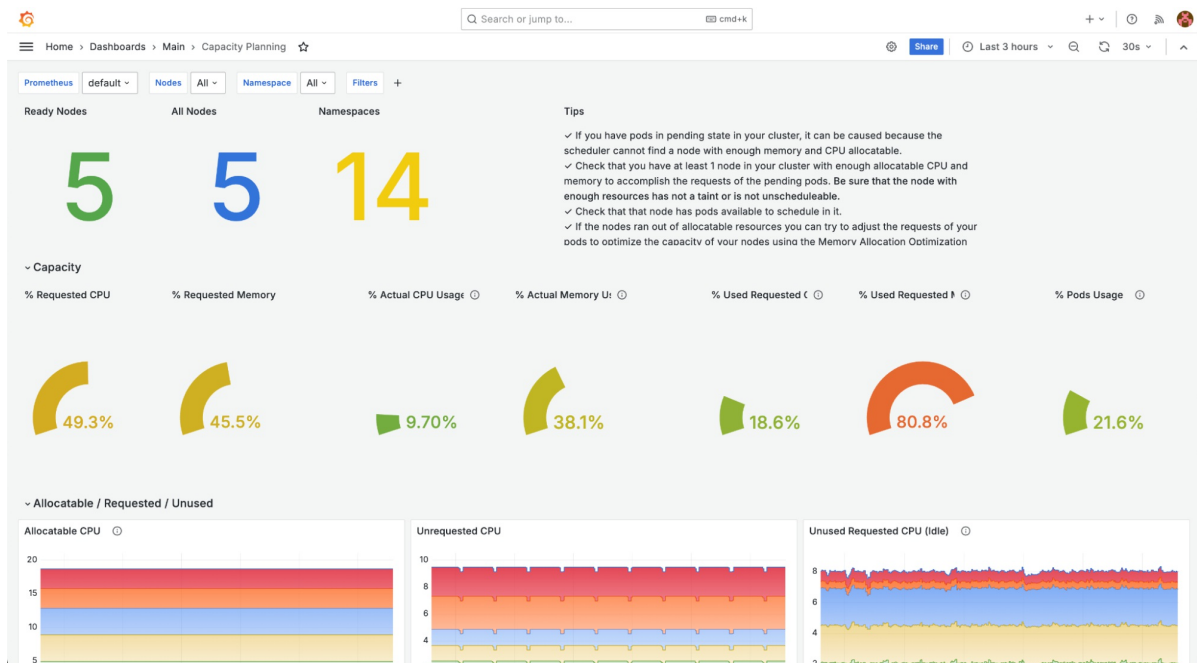
- “Prometheus-(self)” dashboard — summary information about the state of the prometheus module.



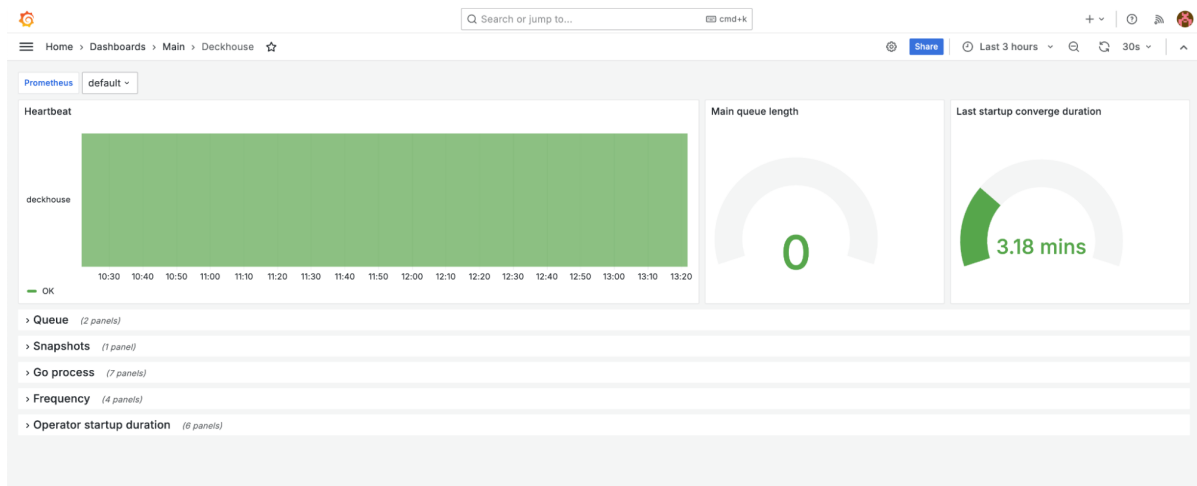
“Main” dashboard group provides general information about the state of the cluster:

- “Capacity Planning” shows overview of the cluster’s performance and capacity usage.

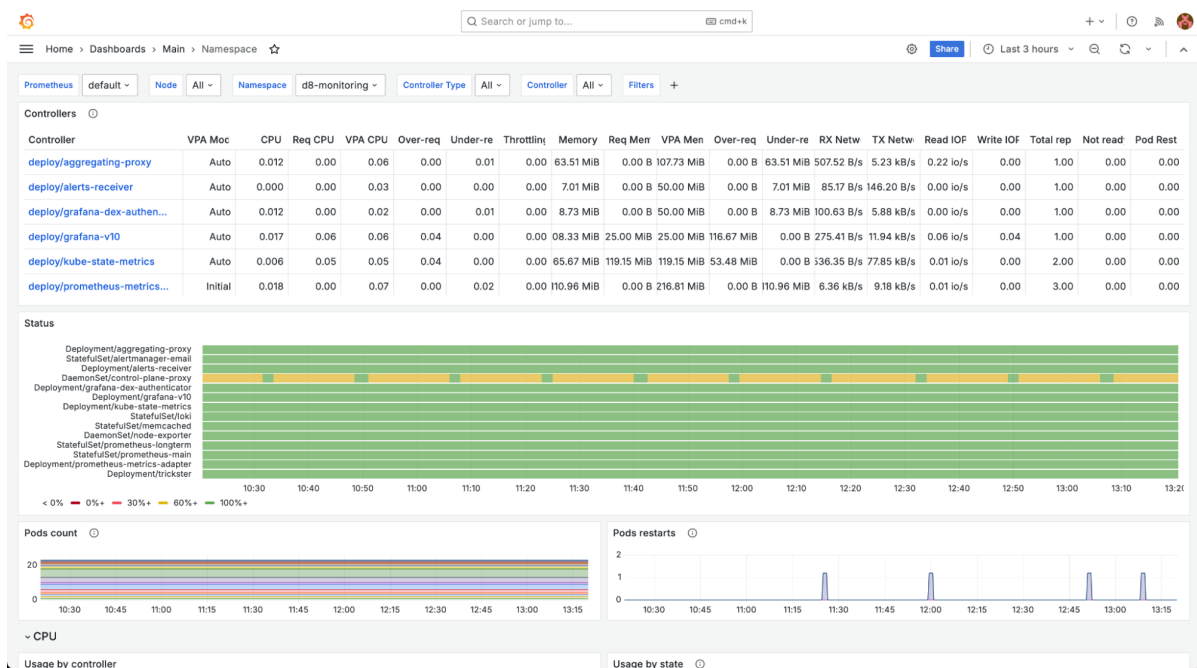




- “Deckhouse” shows summary information about the main deckhouse component.

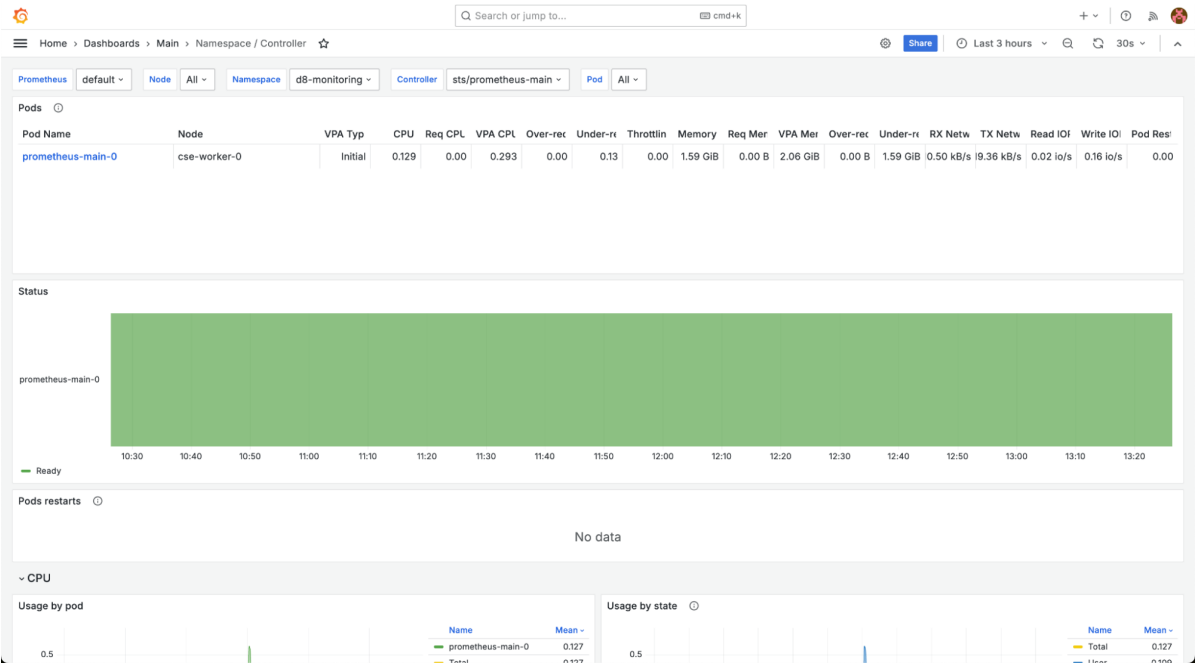


- “Namespace” displays data on a specific cluster namespace.

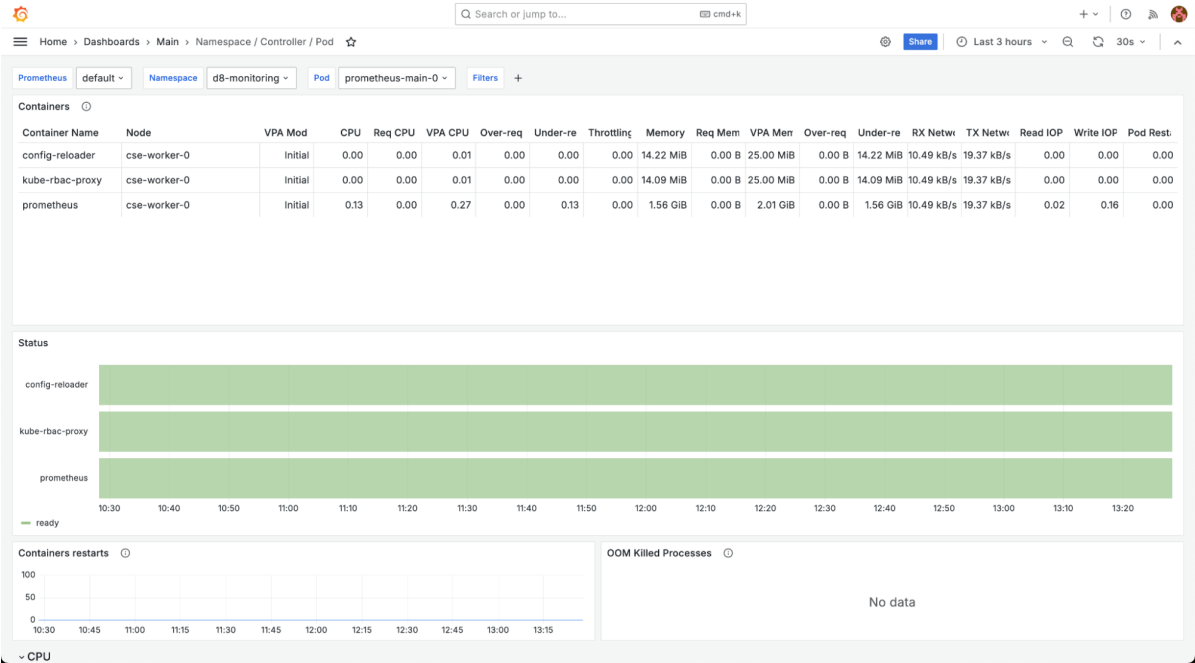


- “Namespace / Controller” displays data on controllers in namespaces. Filters allow you to select specific namespaces and controllers.

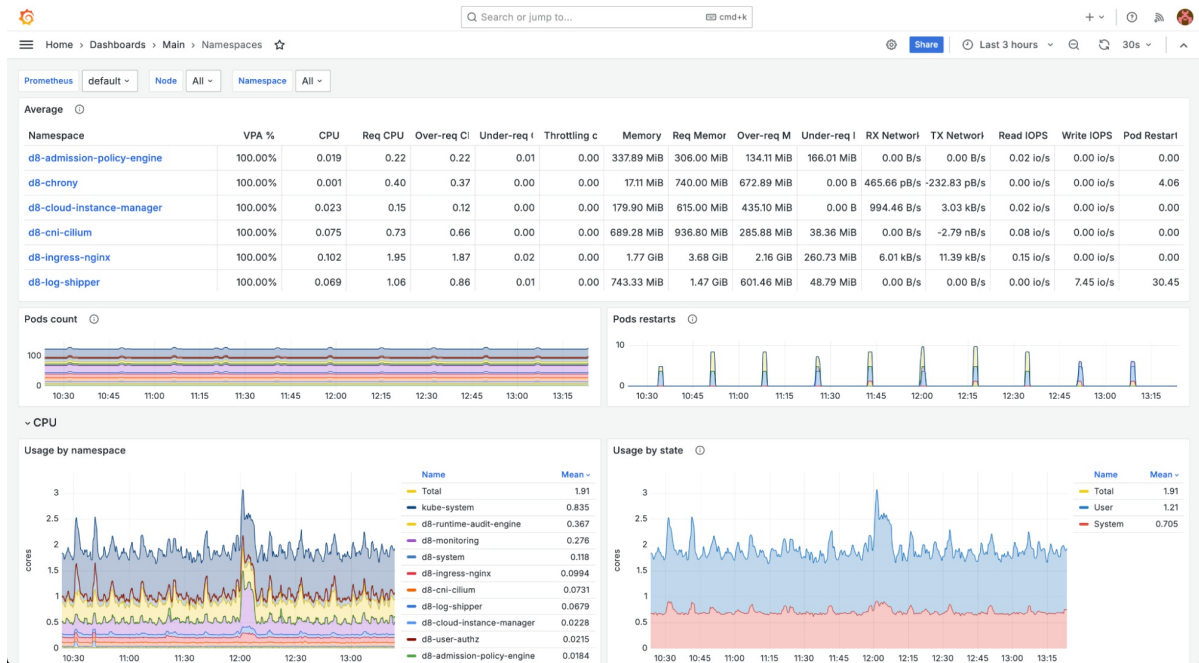




- “Namespace / Controller / Pod” show data on Pods in namespaces. Filters allow selection of specific namespaces and Pods.



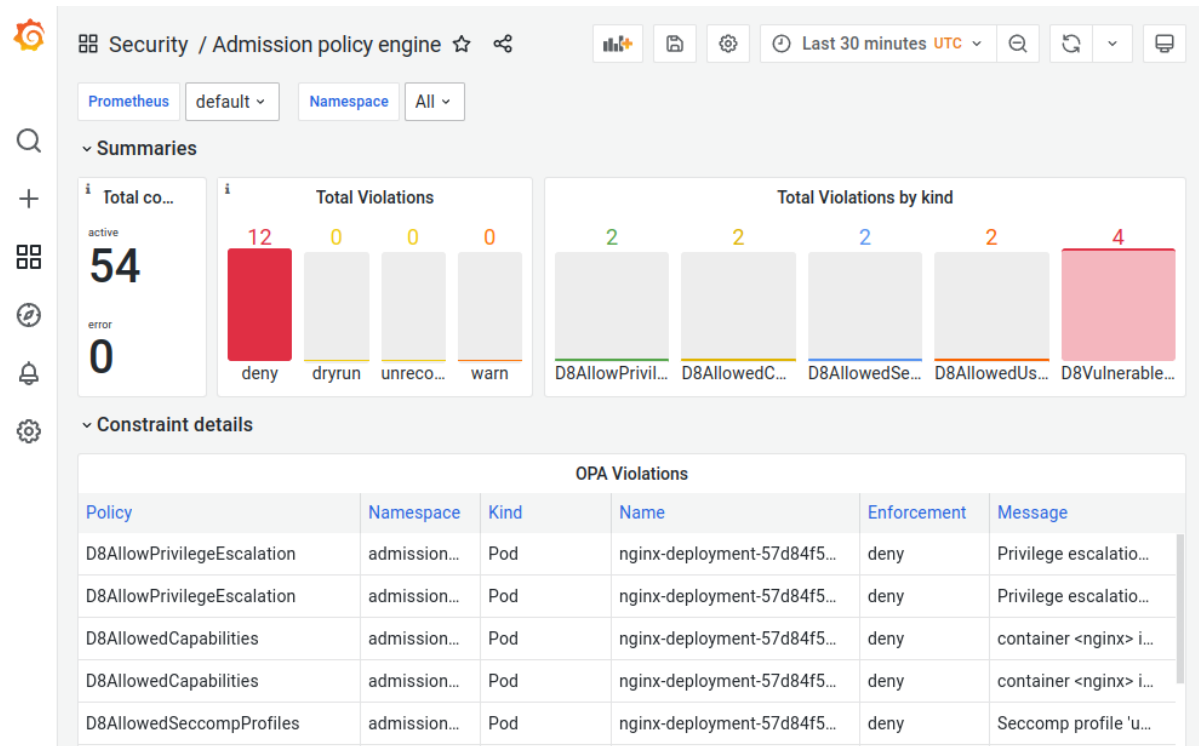
- “Namespaces” shows summary data grouped by cluster namespaces.



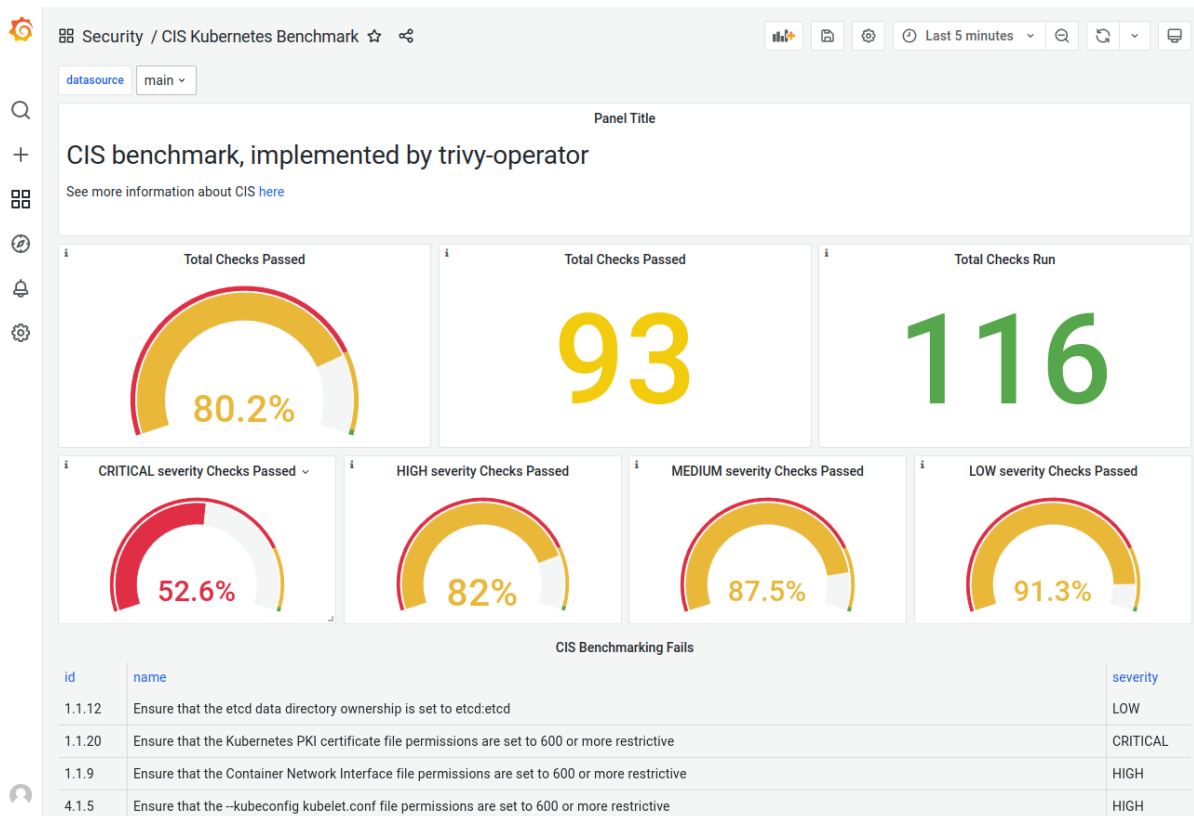
Viewing security event logs

Security event logs are available through the Grafana web UI. The relevant dashboards are grouped under the “Security” folder:

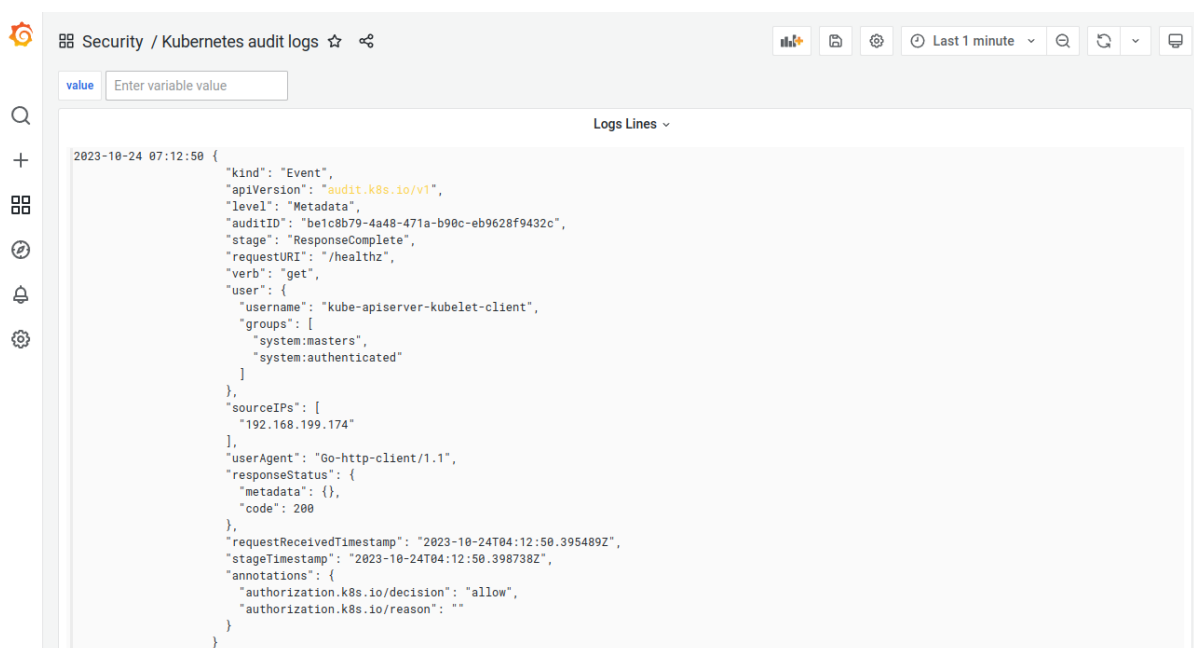
- Admission policy engine. Contains information related to the operation of security policies, including a number of denied actions due to policy violations, breakdown of denials by type, and event logs. The event log for policy-related security violations is available in the OPA Violations panel.



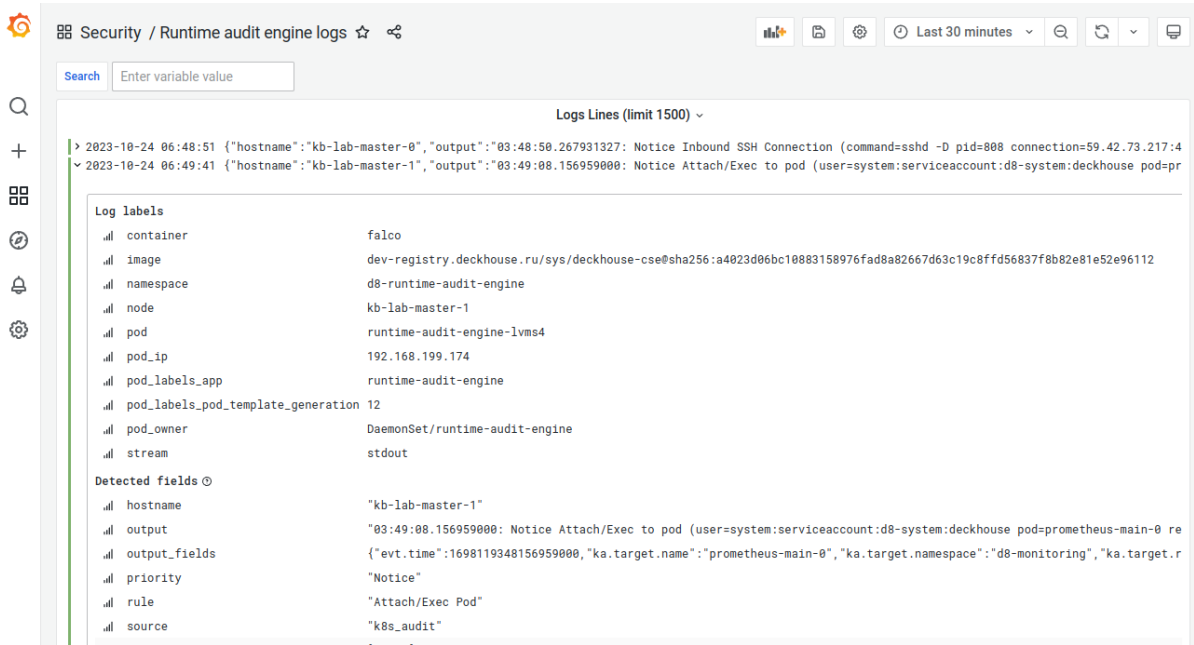
- CIS Kubernetes Benchmark. A dashboard with the results of a configuration audit for compliance with established best practices. It provides summary information with no detailed view. This dashboard is available when the operator-trivy module is enabled.



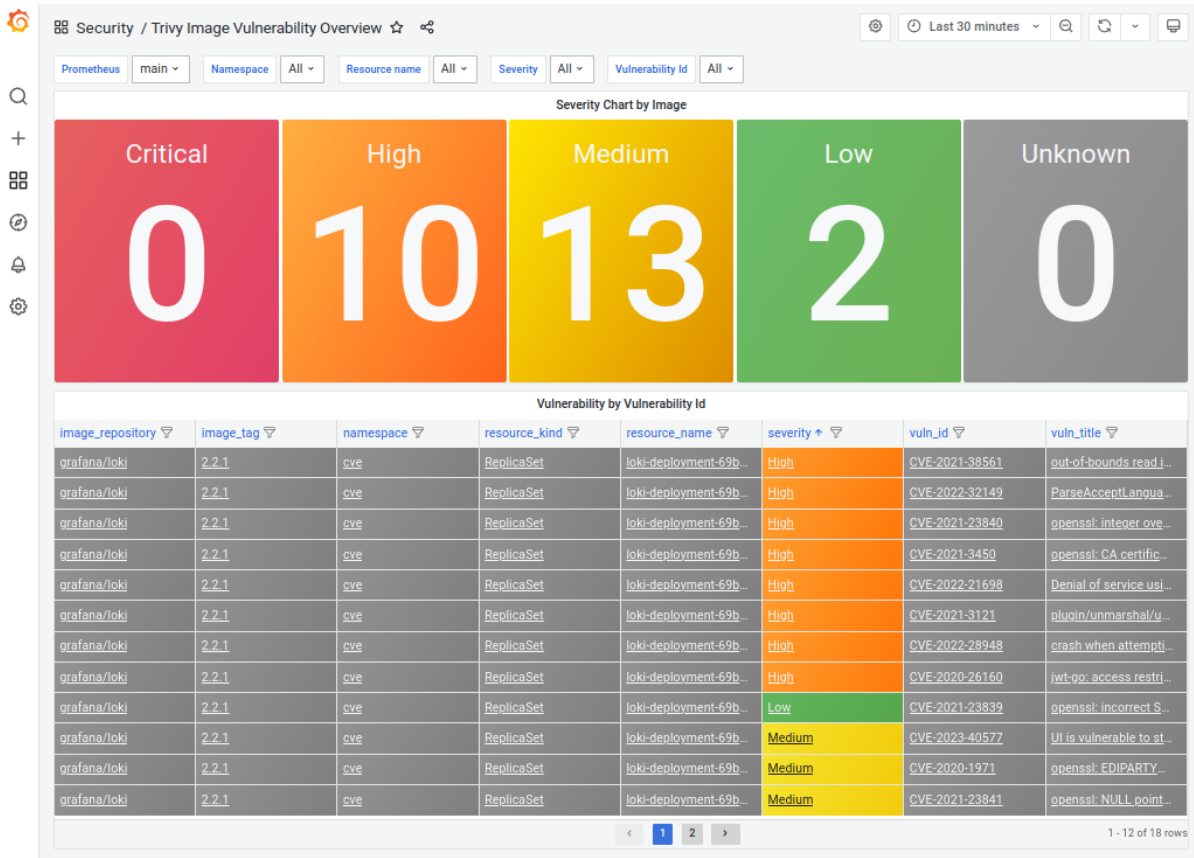
- Kubernetes audit logs. Includes logs of all interactions with the Kubernetes API server. Records are presented in JSON format.



- Runtime audit engine logs. Includes logs of Linux kernel and cluster API server security audit events.



- Trivy Image Vulnerability Overview. A dashboard with summary and detailed information on vulnerability scans of container images in namespaces annotated with `security-scanning.deckhouse.io/enabled`.



Additional resources

- Module `cni-cilium` documentation
- Module `deckhouse` documentation
- Module `log-shipper` documentation
- Module `loki` documentation
- Module `operator-trivy` documentation
- Module `prometheus` documentation

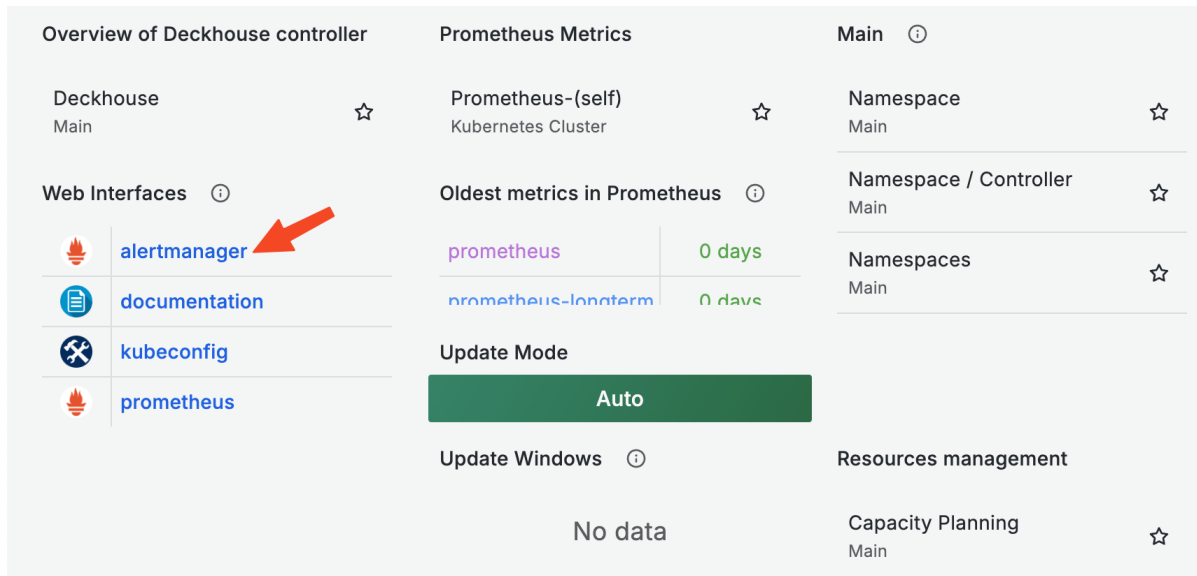
Web interface for managing alerts

The alert management web UI can be used for handling alerts in a Deckhouse cluster. It allows you to view detailed information about alerts, including the source, severity, and trigger time, and to temporarily silence alert notifications when needed.

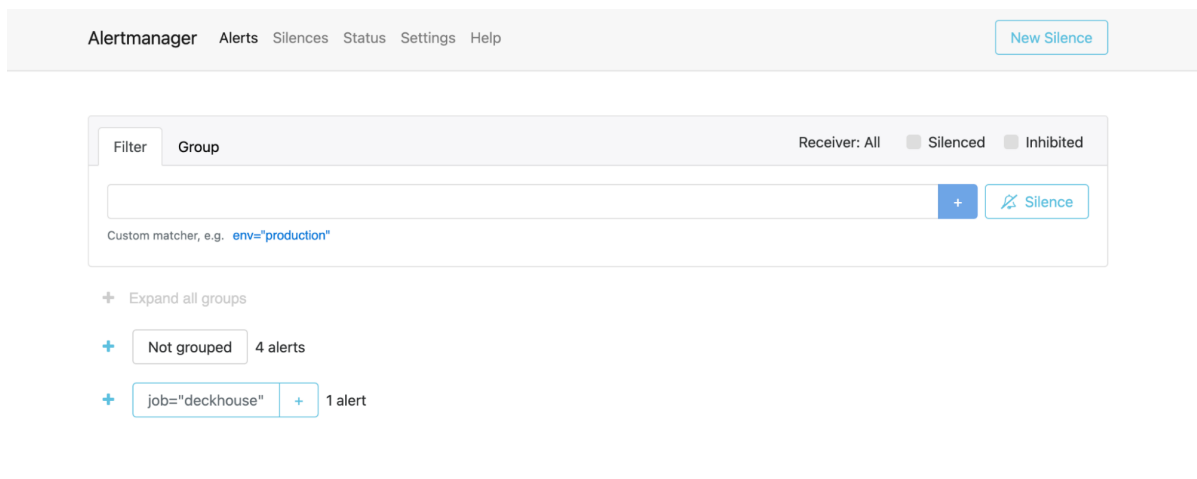
For the list of all available alerts in the Deckhouse Kubernetes Platform monitoring system, refer to the corresponding documentation page.

## Accessing the web UI

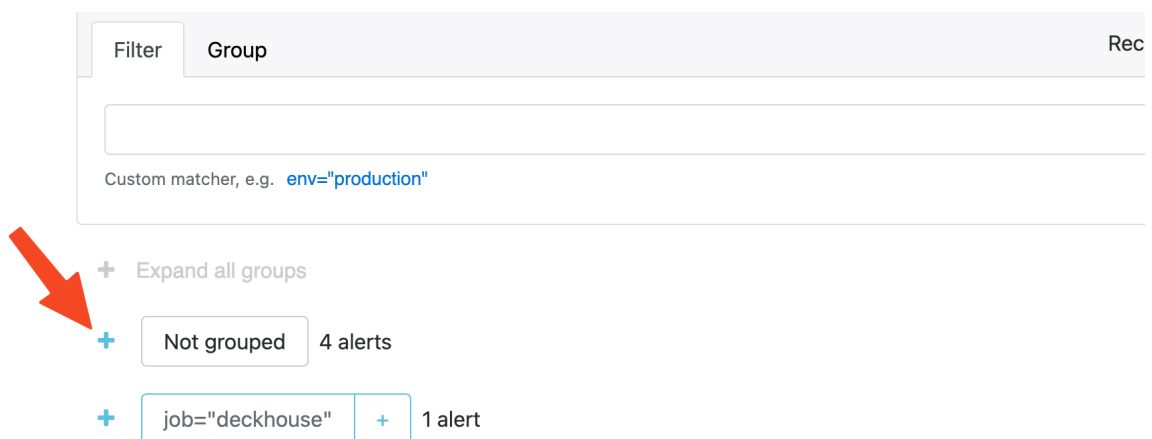
1. To open the alert management web UI, click the corresponding link in the side menu on the Grafana overview page.



2. If you are accessing the web UI for the first time, enter your user credentials. Once the authentication is complete, you will see the main web UI page displaying a summary of all alerts in the cluster.



3. Alerts are grouped by category. To expand a group, click the + icon to the left of the group name.



4. Once expanded, the group will display the full list of associated alerts.

Expand all groups

Not grouped

4 alerts

2024-10-28T10:56:37.543Z

+

 Info

Source

Silence

Link

alertname="DeadMansSwitch"

+

prometheus="deckhouse"

+

severity\_level="4"

+

2024-10-28T10:56:50.558Z

+

 Info

Source

Silence

Link

alertname="D8DeckhouseIsNotOnReleaseChannel"

+

d8\_component="deckhouse"

+

d8\_module="deckhouse"

+

prometheus="deckhouse"

+

severity\_level="9"

+

2024-10-28T11:01:58.385Z

+

 Info

Source

Silence

Link

alertname="D8GrafanaDeploymentReplicasUnavailable"

+

d8\_component="grafana"

+

d8\_module="prometheus"

+

prometheus="deckhouse"

+

severity\_level="6"

+

tier="cluster"

+

2024-10-28T11:01:58.385Z

+

 Info

Source

Silence

Link

alertname="D8GrafanaTargetAbsent"

+

d8\_component="grafana"

+

d8\_module="prometheus"

+

prometheus="deckhouse"

+

severity\_level="6"

+

tier="cluster"

+

## Web interface for generating kubeconfig

The kubeconfig generator web UI can be used to create configuration files required for connecting to the Kubernetes cluster via the `kubectl` tool.

### Accessing the web UI

1. To open the kubeconfig generator web UI, click the corresponding link in the side menu on the Grafana overview page.

Overview of Deckhouse controller

Deckhouse

Main

☆

Web Interfaces ⓘ

alertmanager

documentation

kubeconfig

prometheus

Prometheus Metrics

Prometheus-(self)

Kubernetes Cluster

☆

Oldest metrics in Prometheus ⓘ

prometheus

0 days

prometheus-longterm

0 days

Update Mode

Auto

Update Windows ⓘ

No data

Main ⓘ

Namespace

Main

☆

Namespace / Controller

Main

☆

Namespaces

Main

☆

Resources management

Capacity Planning

Main

☆

2. If you are accessing the web UI for the first time, enter user credentials.
3. Once the authentication is complete, you will see the main documentation page. This page contains configuration parameters for accessing the cluster.
4. Select a tab with your operating system (Linux, macOS, or Windows) in the mid-screen menu bar. Depending on the system, you will see the configuration commands, which will create the required context for connecting to the cluster once they are executed. Also, you can select the **Raw Config** option that allows you to manually add the configuration file to the `kubectl` configuration directory.

Generated Kubernetes Token - https://[redacted]

Login Again

Please check that you have been assigned to the expected Groups, then follow the instructions based on your OS.

```
{
  "iss": "[redacted]",
  "sub": "ChJzZWNIcm10eUB0ZXN0LnRlc3Q5SGxkYXA",
  "aud": [
    "kubernetes",
  ]
}
```



```

    "kubeconfig-generator"
  ],
  "exp": 1730287093,
  "iat": 1730286493,
  "azp": "kubeconfig-generator",
  "at_hash": "HNxfW5-6pY-CgC0JTdZ5_Q",
  "c_hash": "hBbFU1cCs8UvTdFtm_ETQ",
  "email": "security@test.test",
  "email_verified": true,
  "groups": [
    "security"
  ],
  "name": "security@test.test"
}

```

| Kubelogin | Linux | MacOS | Windows | Raw Config | ID Token |
|-----------|-------|-------|---------|------------|----------|
|-----------|-------|-------|---------|------------|----------|

## Kubecttl + Kubelogin

Kubelogin plugin is designed to run as a client-go credential plugin. When you run `kubectl`, kubelogin opens the browser and you can log in to the provider. It also prevents losing refresh tokens and receive errors on concurrent requests.

Install the latest release from Homebrew, Krew, Chocolatey or GitHub Releases.

```
# Github releases (macOS, Linux, Windows and ARM)
github.com/int128/kubelogin/releases
```

```
# Homebrew (macOS and Linux)
brew install int128/kubelogin/kubelogin
```

```
# Krew (macOS, Linux, Windows and ARM)
kubectl krew install oidc-login
```

```
# Chocolatey (Windows)
choco install kubelogin
```

Kubectyl configuration file that you can copy to `$(HOME)/.kube/config`

```
apiVersion: v1
kind: Config
preferences: {}
users:
- name: "client-authentication.k8s.io/v1beta1"
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1beta1
      args:
      - --oidc-login
      - --get-token
      - --oidc-issuer-url="https://kubernetes.default.svc.cluster.local:443/oidc"
      - --oidc-client-id=kubeconfig-generator"
      - --oidc-client-secret=8SYA41ljn76b3XfE55Au"
      - --oidc-extra-scope=email"
      - --oidc-extra-scope=profile"
      - --oidc-extra-scope=groups"
      - --oidc-extra-scope=offline_access"
      - --oidc-extra-scope=audience:server:client_id:uid"
      - --certificate-authority=
data:
- S0tLS1CR1dRZj1TD0RVJ1U2J0QUR5S0tLS0tCk13SUU1ekNDQ0XmR0F3SUJBZ0tDRUFRB0dRRWUwL1p1a3HJzJkFRkRUUxClUUF3RkRfU0t1CQUD
BMVWF0X4d35mEySXQYkdGaUxXtHnNQjRYRFRJME1EUXdNVEV6TVRZek1sb1hEVEKxTURRd0t1URXpNVf16TWxvd0dERVdNQ1FHQTFVRQpBd3d0YTJz
JdGJHRhLWME5sY25abGNgQ0NBUE0t13RFFZSkZkVWk1odMNOQVFQk1JQURnZ0tVQURQ0FRB0Nz0VCkFPNDBKjRE5UXJ0VktWMXf0Y0VR1ERbcwdZV
MVHqRSk4vdmFqUERUNGQ0V0t1BM1dQNMVMSU0y0FI4WjBrZVM3QUKKKc29lbz0Y7Y9aUggrZvdkbmXhZ210NHgYVux1Ky8YUfN00WxLeEM4NDdLa1F
RW1NFRZ1Yk84b1LGTf1aYXUxU0pPY01keG9wTDE0QRjVEJBeDMeCf3UG05RWp13RQ0tJ03tGdEemk10FB1Mz5VmnhZNCX3b3QWctHm1Ph0EQ
y1JL4QkYyc1JL1tqRTYSMJC3tJEvZDRpvn1tdHTak1L3R3AS5UC1dmK9KXc1Rm4GwXhS3BAe1Z2QUDpdmtnb1sK53pctakcXcHFZm3J2NDI2RZh
ubZbHnUpUw1J6T729tCNG6aU53R1kvBGFQUpBt1h6M590dm5QNzN1JDFjYjrcwptUQ15Smf1c1FOWURUW1dQaj1L2dXB10ENB0dVB0WF0Q0FUMHd
nZ0U1TfUR0E8xVWRf1FDUTB0dVRWUUpZ5VpJckFZYjRRZ0VCQKFRREFCNqFNRE1Hq1D0R1NB0RcRU1LCRFRRBzUp1BjR1Z1VTF0TU1FgXz1bVZ
5WvH5BpRk0tQkWLhM1pY5WdRM1Z52Sd5bWFXtMhK1V35FFZRFZ5ME9Cq1LFRKMc3jNMWRdF3Yhdc0t1aU1HdXh5GJ5wTfYApNRTHtQTFVZLc
3UK1NRWfBRK51bghia1SZTFrb3tRSzJ7TmTGM3Xpwb1kxZ295aWtGakFVTJ3d0VBWURUWVFECkRBBHJZaTfZvWdJfKyk0MQRXLFKZ1F0H1ND4np
0Vm1HSTJZ7JfAdYvYHBCU0E0R5vXWRdE0VCL3dRUF35UYk0RBBVEJnT1ZU1VFRERB50cJnZ3CZ0VG01FjREFUQ1BCZ05WSFjFRVNEQkdnaUj
yWwKc1LXSXVabT0kZEHKdgkpQzFwLcxdkxtaGmbVpZwMcXt1ZiWUlpS2k1c1pM0NXZ0t1lWm05NGR1SznKzQzFwLcxdkxtaG1MbVpZC1L
XNTB8M5U2Y1RBTk3Fo2t1H0Xc0wKfRC0ZB0UQ0WdFQXg1MCtB0c1RVfGk0x4MGZHWjFnZ2Vt0ZVMCskT9wYc1LSVzBrd1p2WEZ5wJwNvY2d
GMWtW0G9sSm5E0fnaCfUd1nAkVpMkYrEk1TfJYtBbVZYRFZz3BGUQpEYU1jCENUMd1a1pJNVNFNDFMV3Z05E16VTV2N19VM4E4VfFobmh
YQnLUQ3I55zYKkZvXU1d6t2Nn53RKC1RETUlpnKM4c2t1N1VFM3N0Y0JJEvXWZMm1bF8G0FGUE5xa0xxdWZ50tshaE1tUzCdDvPZFBtM1p
3B6Q520Ug0K1wL1J1NE1GNS2tB38kb3fCtCj0y0HdSTEVDU1vU1vMtnGefPw18rVnZfQ0k45D10NDFTVU5NMV0CkAphZjZXt0NC5k3IvTN
2WtG52tNPdNXTvH0a1EdU5Z0vErbtmhZ1VHL290qTJkV0eRQMwU0L1NXStYkEbfJ0GZxKc1UNTE1RyXtMn5Sm9M01V5008PYjQyNVBEYmQ2N3Z
3TGxAgvG0dEt43jJ0MxRUW1YczbYd1R5GJIRj3C90tJW0a5Esm5vQ0ZRDVFLSK5ZNEy1Y53BoaGtXm5Z50y04kXZm9y00wXBEZT1LRRXU1Y1
M1B2R3ZUJHVVV1Z20p3Y8Y3Ayat21nKzYU7jUhaUfHd3FXNXZVWbZ1UadDUKJZU2yL85rYtK0F1SRWZNM1Bma24y4k1Ud0dCn9WaTe
3T2RwXkXMa3nyT21Scd0p132ZSRXWU1Y2YWM825FVKvNBG5DvWb1Y9uak8WmNUm1E0TG1V1LQKMTdHd1CTHV50xyRdFNKdTRERApU1
kwwFMTGBYd01IDFF0EJCW9aQ3hNQZt1EaG5SMZVzb1p1cFwVXR3aAoxdG1KVDJ0cU400EMrcnc9C10tLS0tRU5E1ENFU1RJrk1UDQVRFL50tLS
k"
```

```

command: kubectl
clusters:
- name:
  cluster:
  server:
  certificate-authority-data:
L50tL31CUDJt1IB0RYJUSUJZ0J00FARStL50tCk1JSUZIENDQXDtZ0F3SUJBZ0LVVfFsMdhmcXpUIE0xv0LZanF0N1LTSG10T2tFd0RRUpLb1pJ
ah2J7kFRRUwKQ1Fbd0ZERNVnQkPHQTFRVUF3d0pHkL0YkdGaUxvTmhNqJrYRFRAJME1UXDNVEV5VYRRek9Gb1hVEv0wTURNegpNREV6TVRRek9G
b3dGREVTYJBR0E0vUUBD3dKYtJYdGJHRmLMV05oTULJQ0lLQ05C22taGtpRzL3MEJBUUVGQkFBT0NBZ3htBTU1JQ0N5NB0ZBVM5tSS5yXWd1J
aFhZm1ckxwPmdnA2R2rdRW5eGdPVL24czlTdnNFU21vd1YKVZV4nCJR5mXpQVB0WmRQ5nQyZDUyZzVYakRNdLE2MGk1kTDYRmNA2M0ZBRUdVn2L
NHW2wlcqWxvBQYvYt3ZTbgtVMEpJNHPR4DNHFYUfQe9aa0Y1S1hHNxLOemc5YmE2d043a1fYtYk0VU5YUUYrdUdt0tL1wa1dYt3YZ3VGN5aTRWC LVk
UVRubXFOeGvMc0t3ZDYzZj1QeD1LVVNj2bF2cEh5dzh5W1ITGpvYzZ1aG0NmM5RG1tNnY4WkU4RXJqMCsKWZ0Uk5yM1ZYnYF6R19S0kxzdG5z
TGTVLn1ycjBcZlYz5UT37j0Y1p5kdybKdLzmtadEN6WfCv2mLLb0V0Tgp0zK1B0XJYNDLub2Jo0G150EtDSUdJN3owQhntS0d3ZfP3XWRBVS5tm
QX1Z2UcyRDJ1YUppGVUNXt2NtH3BXQTh0cK50eG1ZdF0ZMkxa0EFYbYhYUuK2V0xSnmMwZVJVJ0Y56cDhCRU9ybUj1eGF5UHHMvNFWlnFSU2RNM2Jo
Z1h1VncK5URZWHZ5YtBM2Q4NMh5v2Sw1LhwRE55Y9tCuzTSKRXVmhYUdCMWPVZ3NIRE1KfYK5bGqRcLTlTFRNWG1a0pFMVB4ed1RQ0xENNVF
ZVfCjHkVbXg10Y84UXfIvmtXhWz5ZU1TlPc1mNvZVz52YdZtQKRBZ2V9K5SE1pY1BqCf13YhBvZ3UDJ3FOEYFCG52adNveEpUkU6E2Zkx0cU
Wdjdjb2b0kN28ZRFYc4EhRhnaZmF1TtYpV5VROZVYtW82bUjV0mQ5ND0dR1p1bYhzb2pMEF9303R6ZGD32mP5N1Em8r0XV5z20Vb1hJnHFW
0kzhWkZmZmJLBNQw0bD0BWF0bH1UHXdlUVEVL1wT0JCMUVGT3vS4G3qk1JlMwtvctLML10ZDcxenbVfTnTU14R0E5WmR3d1FZCk1CYUFG
Tn5vAg3K1JlMwtvctLML10ZDcxenbVfTnTU14R0E5WmR3d1FZCk1CYUFGTn5vAg3K1JlMwtvctLML10ZDcxenbVfTnTU14R0E5WmR3d1FZCk1CYUFG
U0L1M0RRRUJdd1VGNRakNSNDZ40LdcCjM0SDRUKZ2TNZ3Lw1q5vRja0J0MULJYVniQkU1U7qYE90c1lPvN100BhbTlRMUwzMHhXUHCnJ1Cvdi
MGdxK0Wt0pNslPvGNRakNSNDZ40LdcCjM0SDRUKZ2TNZ3Lw1q5vRja0J0MULJYVniQkU1U7qYE90c1lPvN100BhbTlRMUwzMHhXUHCnJ1Cvdi
d0kM0tNPNVvwx30TFzcwQ4r1dVnGt4ekdwFQRvTEyUdL15LzMH9MUdMwRFBFMDNM1hLa08YyJlBPQ5xMDhPw0wzTE9YUwQwZdLpUC9Xy1s1
CtYkN08ZmHmRZJG50FmdFAl1TmadEXVHNWML29RZEdR5w1UvF0STNCmWKKVeURKQ1J6QhGVK50eKqYRnLaZw5UwNwSsZJ0U1RWTvSN3dZ0HVB
FynF1w90Z1VLvbkJp6SjY1A3YkUw1UBtMuxdYhVMKDYVRtU5X4dZ5mAnR1J3A0K3L0WEF4M1m089E1ZVHRENGc1UVK3Z1ZrYbH6edJqbk9xUfUW
TzhYV53dmRkZpVYJ1J2TA1Z1dDMTtFS1Jb3m9h8XZ0dWkhdhFvYUlnZWhJUB2LZXZ2PSLZFT2CgRqVhpbV1wC0tCMNMNF1LWEN1M
J3WxJg4dKvQvYfIwEN1bz0g0v2YzFVZTJYhZM2FSU1eGpMEHAYvtp0tW5ApU1QvRtGKnLYFXxUUBZ0a0PTRBZ2XUHF4YbA05kNc
dZTUIE1mnYrMfmNMNDM3jYd0A1azdkTzdsV5w0LfnPQ59pTQZ0KRPKQZ5Xv1tFCgw5KvYn0Y0vJTRt0dZNM1dhjYwMz1YfE2U3ZRM3ZHVVFY
QWtTdkRQZ1jQVNGVZn0UKrtCnYek4Zt4RHFJRXNBh1JLJ05SUQTZ72C09CQ0tL50TUVRCRBDR0VJTRUJdQKUR50tL1wa1dYt3YZ3VGN5aTRWC LVk
contexts:
- context:
  cluster:
  user:
  name:
current-context:

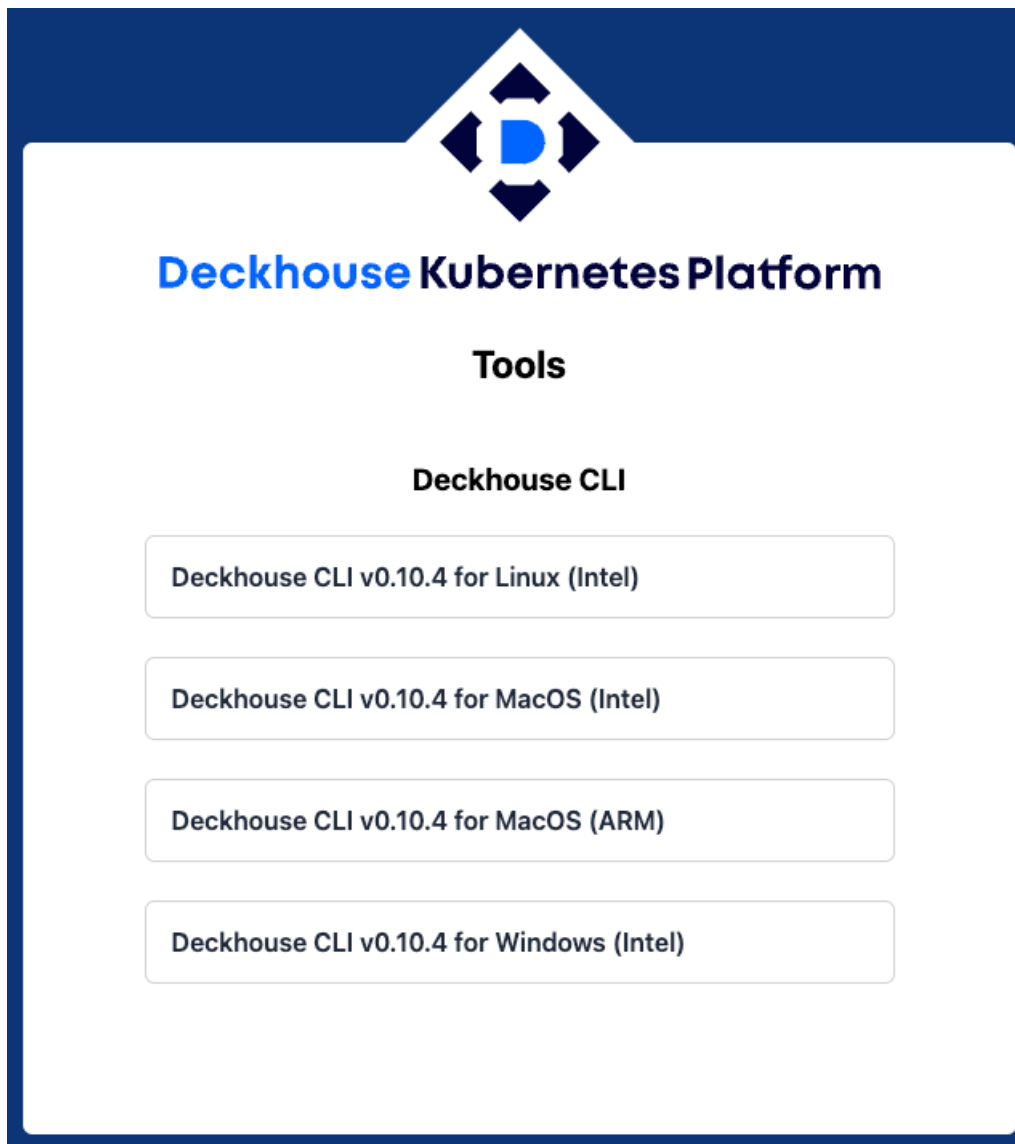
```

## Web interface for downloading CLI utilities

Web interface for downloading Deckhouse utilities provides centralized access to CLI tools for popular operating systems (Linux, macOS, Windows). Using it, you can quickly locate and download executable binaries for the required architecture and version, as well as receive notifications about their updates. This functionality is implemented by the `deckhouse-tools` module.

### Accessing the web interface

1. To open the web interface, enter `tools.<CLUSTER_NAME_TEMPLATE>` in your browser's address bar, where `<CLUSTER_NAME_TEMPLATE>` is the string matching the cluster's DNS name template, as specified by the global parameter `modules.publicDomainTemplate`. The exact URL format may vary depending on system configuration. Check with your administrator for the correct URL.
2. On first login, enter your user credentials. After successful authentication, the Deckhouse Tools download page will open, presenting links to download the Deckhouse CLI tool for various operating systems: Linux, macOS, and Windows.



3. Select the required version, download, and install the corresponding executable file.

### Additional resources

- Module `deckhouse-tools` documentation

## Secret management web interface

The web interface for managing secrets (Stronghold web interface) is designed for managing secret mechanisms, authentication, and access control in a cluster. Operation is provided by the `stronghold` module.



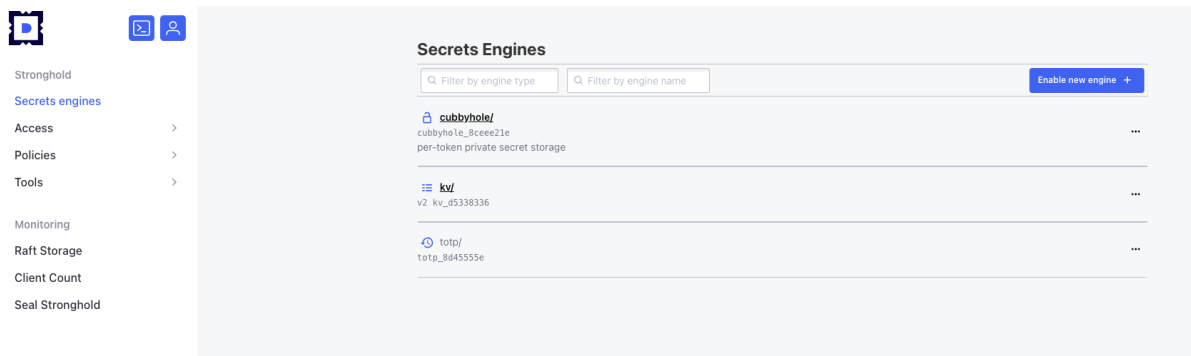
It allows you to:

- View and configure secrets engines, add and edit secrets.
- Manage authentication methods, user groups, and entities.
- Control permissions, leases, and security policies.

## Accessing the stronghold web UI

To open the web UI, enter `stronghold.<CLUSTER_NAME_TEMPLATE>` in your browser's address bar, where `<CLUSTER_NAME_TEMPLATE>` is the DNS name template of the cluster defined in the global `modules.publicDomainTemplate` parameter.

1. On your first login, enter the user credentials.
2. After successful authentication, the main stronghold page will open:



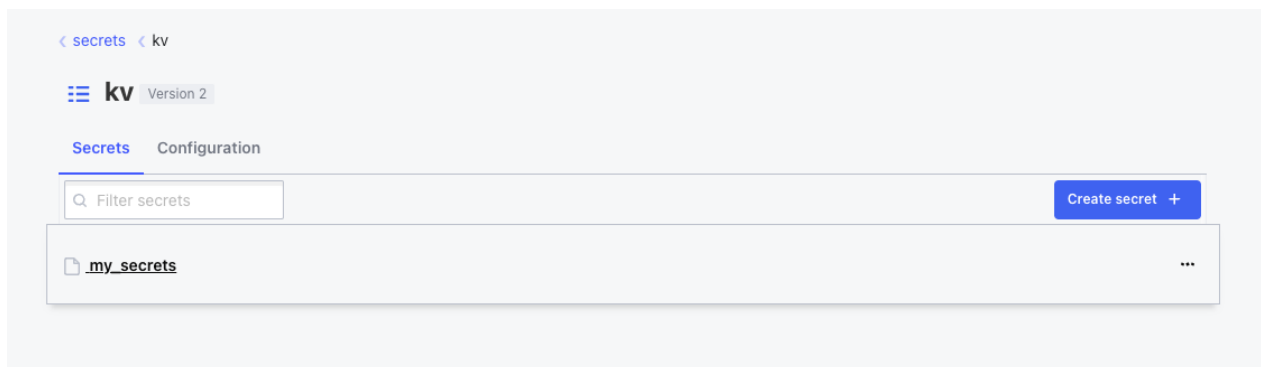
## Managing secrets engines

Configuration guides for common engine types are available in the Deckhouse Stronghold documentation: KV secrets engine and PKI secrets engine.

### Viewing a secrets engine

To view a secrets engine:

1. Click on its name.
2. The following tabs will be displayed:
  - “Secrets” with a list of secrets (roles, keys, etc.).
  - “Configuration” with engine configuration.
  - A button to add a new secret.



To view the configuration of an engine, go to the “Configuration” tab. The content of this tab depends on the selected engine.

< secrets < kv

**kv** Version 2

Secrets **Configuration**

|                            |                                        |
|----------------------------|----------------------------------------|
| Type                       | kv                                     |
| Path                       | kv/                                    |
| Accessor                   | kv_d5338336                            |
| Local                      | <input checked="" type="checkbox"/> No |
| Seal wrap                  | <input checked="" type="checkbox"/> No |
| Default Lease TTL          | 0                                      |
| Max Lease TTL              | 0                                      |
| Version                    | 2                                      |
| Require Check and Set      | False                                  |
| Automate secret deletion   | Never delete                           |
| Maximum number of versions | Not set                                |

To view secret information and versions (for example, for a “key-value” engine):

1. Click the secret’s name in the list.
2. You will see the following tabs:
  - “Secret” with general information about the secret and version history.
  - “Metadata” with metadata of the secret.

< kv < my\_secrets

**my\_secrets**

Secret Metadata

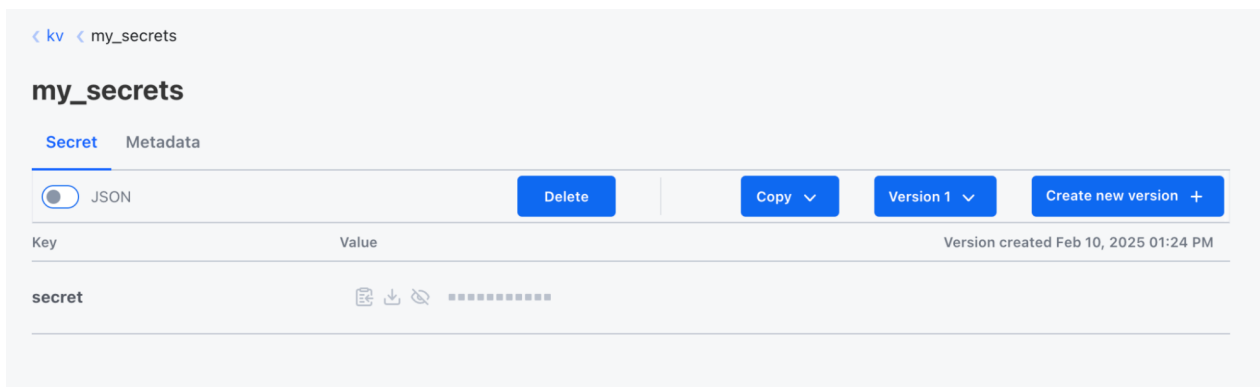
☒ JSON ☐ YAML ☐ Text

Version Data

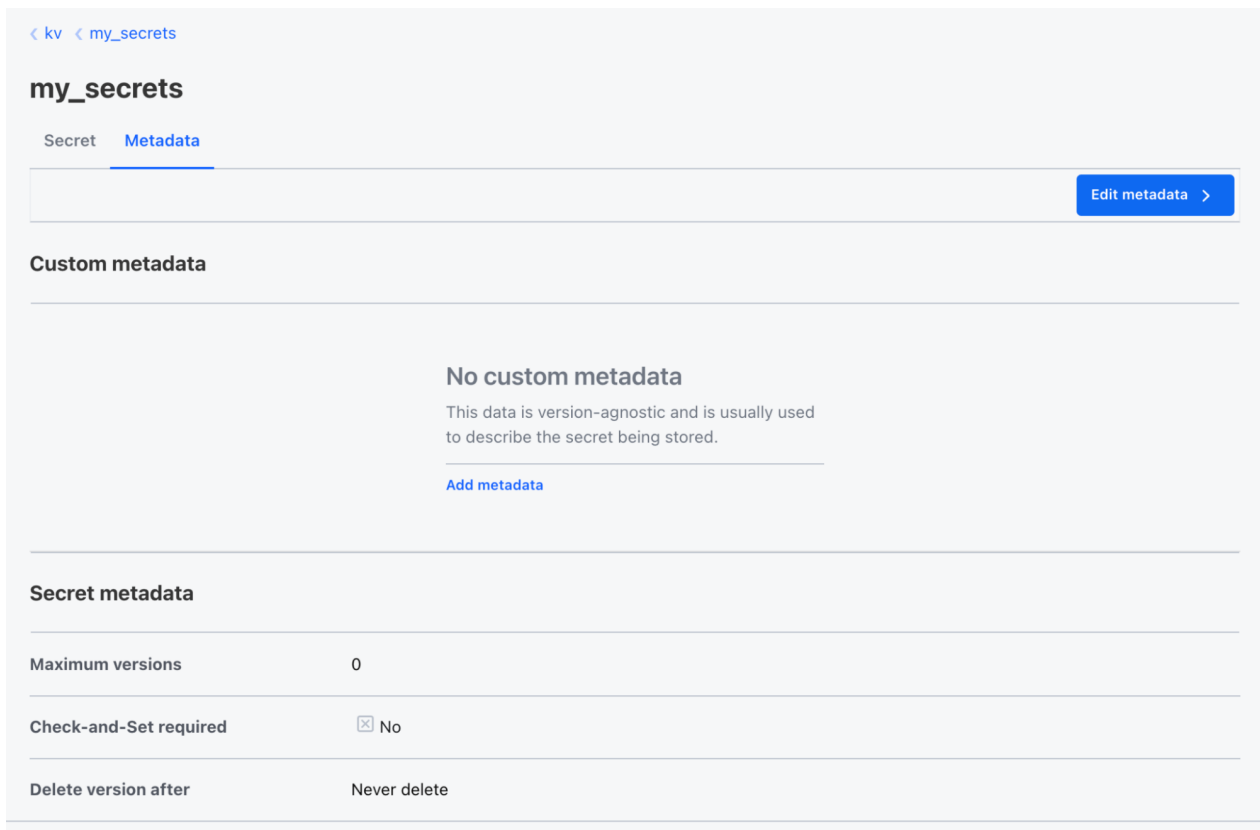
```
{
  "secret": "1234"
}
```

The “Secret” tab includes:

- A toggle to switch to JSON view.
- Buttons for:
  - Deleting the secret.
  - Copying the secret.
  - Selecting a version (if available).
  - Adding a new version.

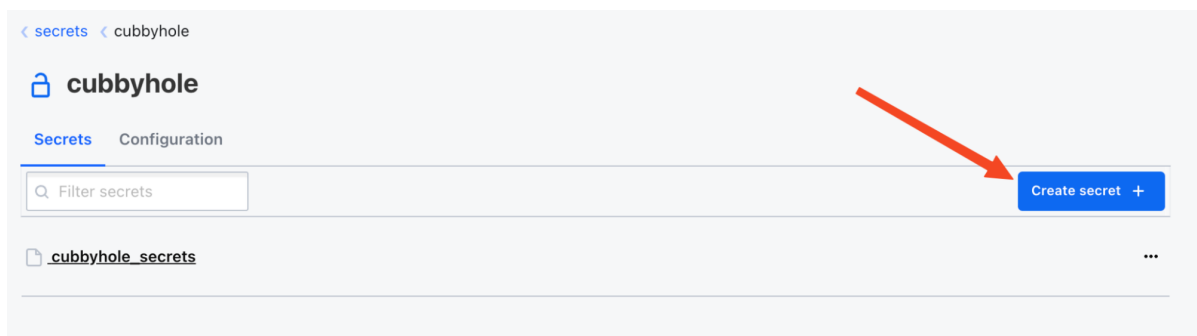


To view metadata, switch to the “Metadata” tab.



To add a new secret, follow these steps:

1. Click the button for adding a secret (button may vary depending on the engine: role, key, etc.).



2. In the form that opens, provide the required parameters (fields vary by engine). For example, the form in a “Cubbyhole” engine includes the following fields:
  - A JSON view and edit toggle.
  - A path field.
  - A key field.
  - A value field.
  - A button to add additional key-value pairs under the same path.
3. Click **Save** to finish.

[← cubbyhole](#)

## Create Secret

☒

JSON

**Path for this secret**

**Secret data**

key

Save

Cancel

Add

To add a new secrets engine:

1. Click **Enable new engine**.

## Enable a Secrets Engine

Generic

|                                                                                                                             |                                                                                                                                |                                                                                                                   |                                                                                                                       |                                                                                                                    |                                                                                                                           |
|-----------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <br>KV<br><input checked="" type="radio"/> | <br>PKI Certificates<br><input type="radio"/> | <br>SSH<br><input type="radio"/> | <br>Transit<br><input type="radio"/> | <br>TOTP<br><input type="radio"/> | <br>Kubernetes<br><input type="radio"/> |
|-----------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|

Infra

|                                                                                                                       |                                                                                                                          |                                                                                                                          |
|-----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <br>Consul<br><input type="radio"/> | <br>Databases<br><input type="radio"/> | <br>RabbitMQ<br><input type="radio"/> |
|-----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|

[Next](#)

2. Choose an engine type and click **Next**.
3. In the setup window, provide:
  - Basic engine configuration (depends on type)
  - Optional method settings (click the header to expand)
4. Click **Enable Engine** to save the engine. Alternatively, click **Back** to cancel and return.

## Enable a Secrets Engine

### Path

### Maximum number of versions

The number of versions to keep per key. Once the number of keys exceeds the maximum number set here, the oldest version will be permanently deleted. This value applies to all keys, but a key's metadata settings can overwrite this value. When 0 is used or the value is unset, Stronghold will keep 10 versions.

### ☐ Require Check and Set

If checked, all keys will require the cas parameter to be set on all write requests. A key's metadata settings can overwrite this value.

### ☒ Automate secret deletion

A secret's version must be manually deleted.

^ Hide Method Options

### Version ⓘ

### Description

### ☐ List method when unauthenticated

### ☐ Local ⓘ

### ☐ Seal wrap ⓘ

### ☒ Default Lease TTL

Lease will expire after

seconds

### ☐ Max Lease TTL

Stronghold will use the default lease duration.

### Allowed managed keys

Add one item per row.

Add

### Request keys excluded from HMACing in audit ⓘ

Add one item per row.

Add

### Response keys excluded from HMACing in audit ⓘ

Add one item per row.

Add

### Allowed passthrough request headers ⓘ

Add one item per row.

Add

### Allowed response headers ⓘ

Add one item per row.

Add

Enable Engine

Back

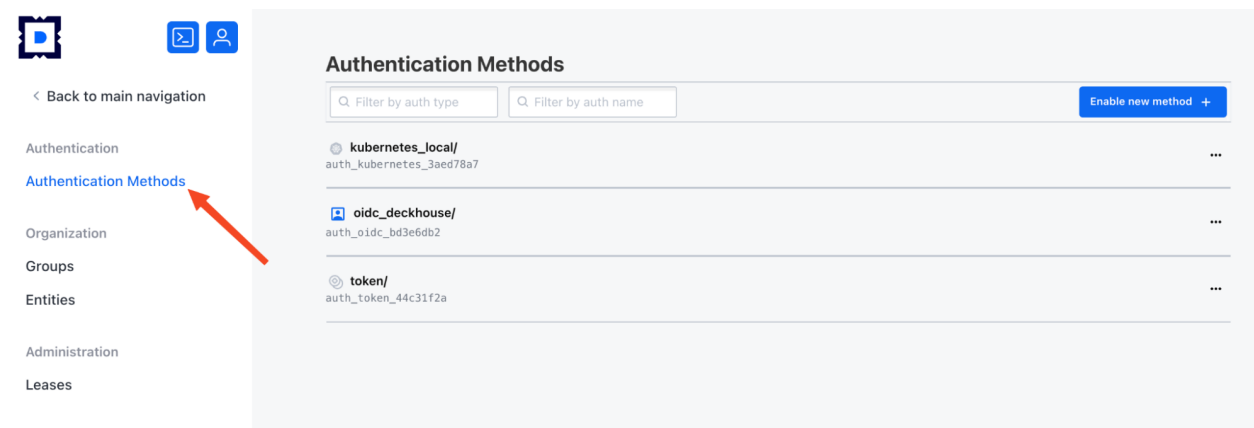
## Managing access to data and stronghold features

Access control is managed in the “Access” section.

To open it, click **Access** in on the main page of stronghold web UI. The left pane contains a navigation menu with a link to return to the main page. The central pane displays content for the selected subsection (by default it's "Authentication Methods").

## Working with authentication methods

This subsection opens by default when entering the "Access" section. You can also click "Authentication Methods" in the left menu.



This subsection shows:

- A list of authentication methods configured in the cluster.
- Filters for the list.
- A button to enable new method.

Available actions for each method:

- View configuration.
- Edit configuration.
- Delete method.

To view method details, click its name or select "View configuration". A configuration tab will appear with a button to configure the method.

For example, for the `oidc_deckhouse` method, the "Configuration" tab and a "Configure" button are shown.

To add a new authentication method:

1. Click **Enable new method** in the "Authentication Methods" section.
2. Choose a method from the list and click **Next**.
3. In the opened window, provide:
  - Path
  - Optional method options (click the header to expand)
4. Click **Enable Method** to finish. Alternatively, Click **Back** to cancel and return.

## User groups

To manage user groups, click **Groups** in the side menu.

This subsection provides:

- A list of user groups in the cluster.
- Filters for the list.
- A button to add a new group.

Available actions for listed groups:

- View group details.
- Edit group settings.
- Delete a group.

To view group details, click its name or select **Details**.

To add a new user group:

1. Click **Create group** in the "Groups" section.
2. Fill out the group creation form.
3. Click **Create** to save. Alternatively, click **Back** to cancel and return.

## Entities and aliases

Entities in stronghold are logical representations of users or applications. They allow linking multiple authentication methods under a single identity.

To manage entities and aliases, select **Entities** from the side menu of the “Access” section.

The center panel has two tabs:

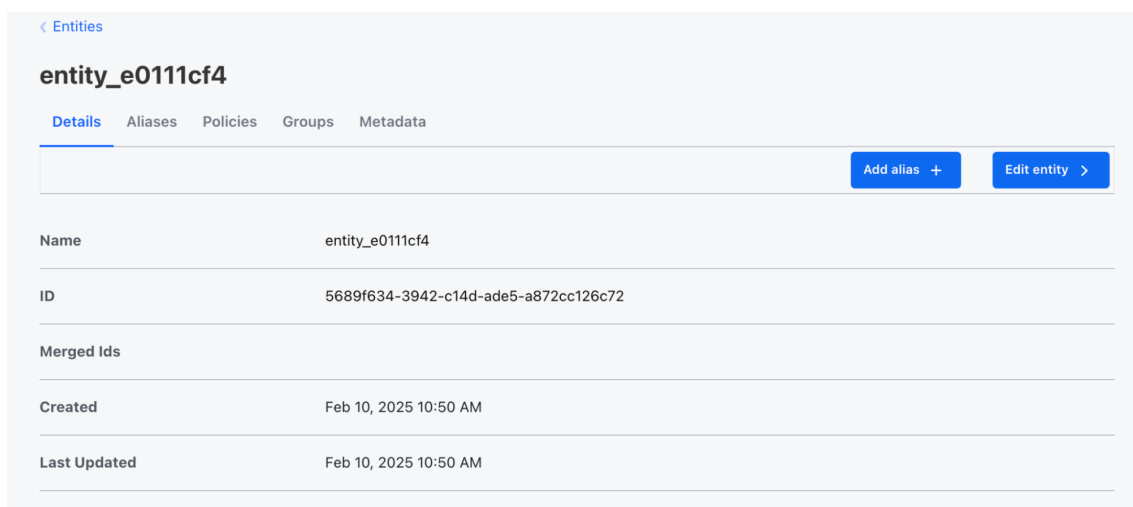
- **Entities** with a list of entities.
- **Aliases** with a list of aliases.

Also available:

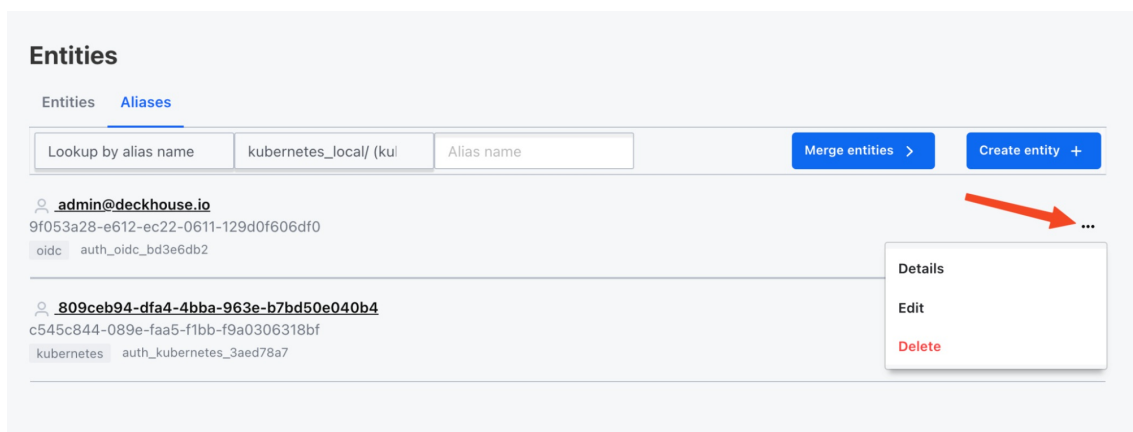
- Filters.
- **Merge entities** button to combine identities.
- **Create entity** button.

Available actions:

- For entities:
  - View details.



- Create an entity.



- Delete an entity.

### Merge Entities

**Warning**  
Metadata on merged entities is not preserved, you will need to recreate it on the entity you merge to.

**Entity to merge to**

**Entities to merge from**  
Add one item per row.

Add

☐ Keep MFA secrets from the "to" entity if there are merge conflicts

Save
Cancel

- For Aliases:
  - View details.

[< Entity aliases](#)

### admin@deckhouse.io

[Details](#) [Metadata](#)

Edit entity alias >

|                     |                                                                                                                   |
|---------------------|-------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>         | admin@deckhouse.io                                                                                                |
| <b>ID</b>           | 9f053a28-e612-ec22-0611-129d0f606df0                                                                              |
| <b>Entity ID</b>    | 5689f634-3942-c14d-ade5-a872cc126c72                                                                              |
| <b>Mount</b>        | auth/oidc_deckhouse/<br><span style="background-color: #f0f0f0; padding: 2px 5px;">oidc</span> auth_oidc_bd3e6db2 |
| <b>Created</b>      | Feb 10, 2025 10:50 AM                                                                                             |
| <b>Last Updated</b> | Feb 10, 2025 10:50 AM                                                                                             |

- Edit an alias.

### Create Entity Alias for 5689f634-3942-c14d-ade5-a872cc126c72

**Name**

**Auth Backend**

Create
Cancel

- Create an alias.
- Delete an alias.

## Managing temporary access to secrets and resources (leases)

To manage leases (temporary access to secrets and resources), click **Leases**. A search box will open that lets you find lease details by ID.



## Lookup a Lease

Lease ID

If you know the id of a lease, enter it above to lookup details of the lease.

Lookup

## Managing access control policies

To manage access control policies in stronghold, click **Policies** from the “Access” menu. The left pane shows navigation menu. The central pane contains a list of policies, a search filter, and a button to add a new policy.

For actions with a policy, use the three-dot menu at the end of the row with the policy’s name.

Available actions for listed policies:

- **View policy details.** To view information about a policy, click its name or select “Details”. The policy details window shows information in HCL format, a button to download the data, and a button to edit the policy.

The screenshot shows the 'ACL policies' section in the Deckhouse web interface. The title is 'deckhouse\_administrators'. Below the title is a search bar and two buttons: 'Download policy >' and 'Edit policy >'. The main content area is titled 'Policy (hcl format)' and contains a code editor with the following HCL configuration:

```
path "*" { capabilities = ["create", "read", "update", "delete", "list", "patch", "sudo"]}
```

- **Edit policy.** To add a policy, click **Create ACL policy** and enter its name and HCL configuration.

[< ACL Policies](#)

## Create ACL Policy

Name

Policy Upload file

1

You can use Alt+Tab (Option+Tab on MacOS) in the code editor to skip to the next field. See [example template](#).

Create policy Cancel

- **Delete policy.**

## Using additional tools

The “Tools” section provides access to extra tools in stronghold. The left pane displays tool navigation. The center shows fields for the selected tool.

Available tools:

- **Wrap:** Creates a wrapping token for secure transfer of confidential data and secrets. This token can be transferred to another user or an application for subsequent unwrapping and accessing the data.






[< Back to main navigation](#)

Tools

**Wrap**

Lookup

Unwrap

Rewrap

Random

Hash

API Explorer

### Wrap Data

Data to wrap (json-formatted)

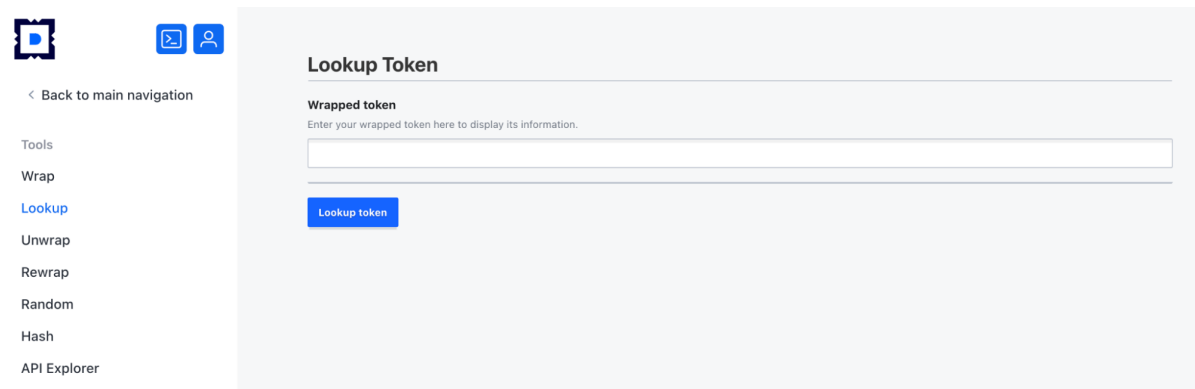
1 {

2 }

☒ **Wrap TTL**  
Stronghold will use the default (30m)

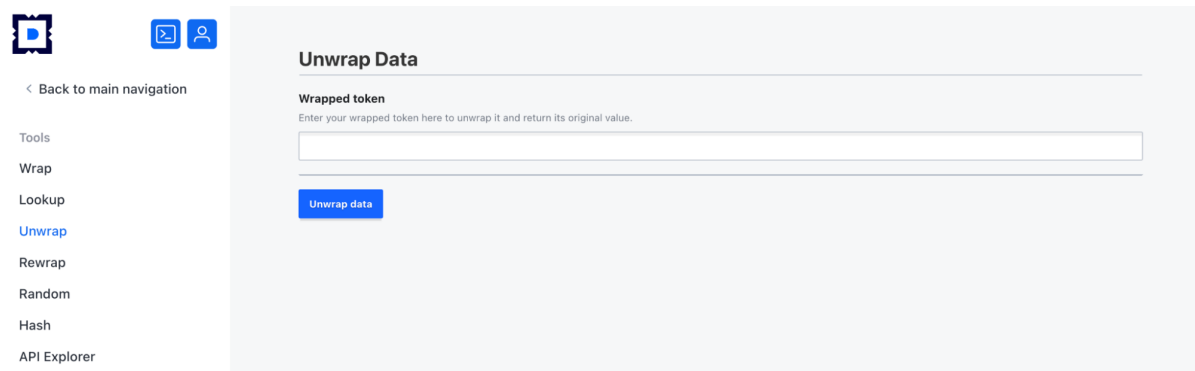
Wrap data

- **Lookup:** Displays details about tokens, secrets, leases, and other objects in stronghold. It can be used to view metadata, expiration dates, access policies, and other details associated with objects.



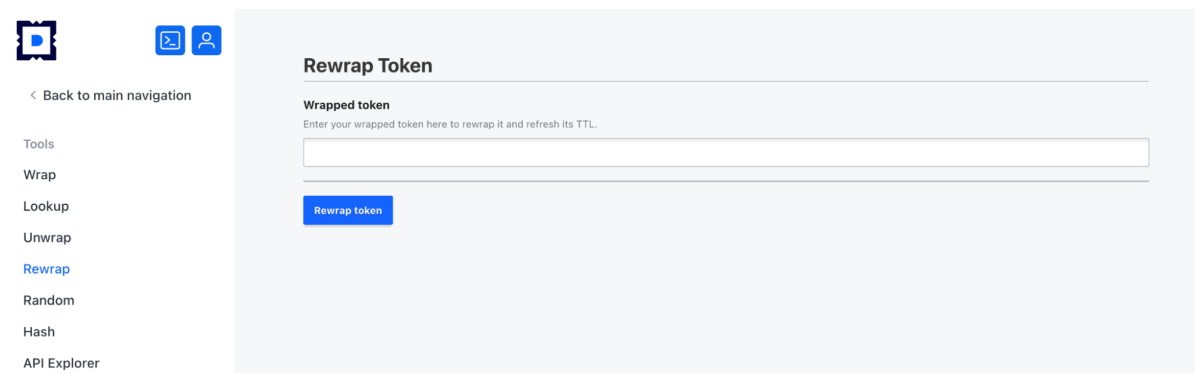
The screenshot shows the Deckhouse web interface. On the left is a navigation sidebar with a logo at the top, followed by a 'Back to main navigation' link, and a list of tools: Tools, Wrap, Lookup (highlighted in blue), Unwrap, Rewrap, Random, Hash, and API Explorer. The main content area is titled 'Lookup Token'. It contains a section 'Wrapped token' with the instruction 'Enter your wrapped token here to display its information.' Below this is a text input field and a blue button labeled 'Lookup token'.

- **Unwrap:** Unwraps a wrapping token to access the wrapped data.



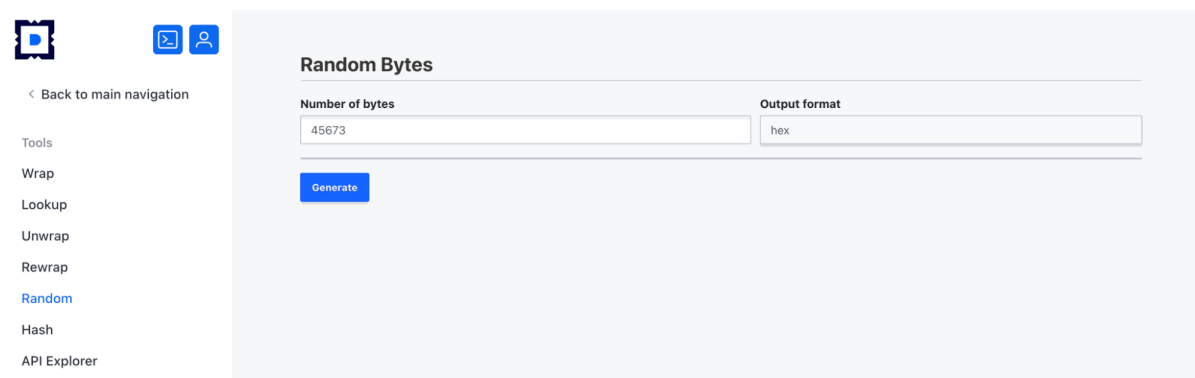
The screenshot shows the Deckhouse web interface. The left sidebar is identical to the previous one, with 'Unwrap' highlighted in blue. The main content area is titled 'Unwrap Data'. It contains a section 'Wrapped token' with the instruction 'Enter your wrapped token here to unwrap it and return its original value.' Below this is a text input field and a blue button labeled 'Unwrap data'.

- **Rewrap:** Reissues a wrapping token based on an existing one. This lets you extend the token's TTL or modify its parameters without disclosing protected data.



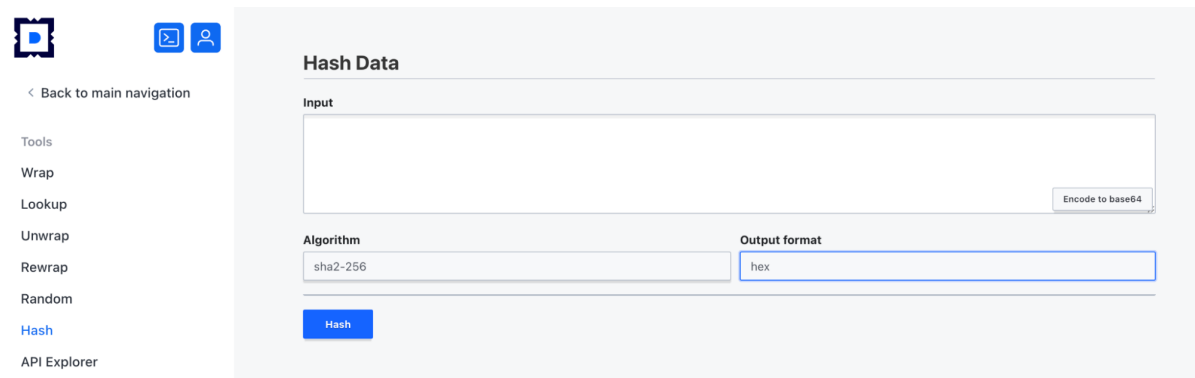
The screenshot shows the Deckhouse web interface. The left sidebar is identical to the previous ones, with 'Rewrap' highlighted in blue. The main content area is titled 'Rewrap Token'. It contains a section 'Wrapped token' with the instruction 'Enter your wrapped token here to rewrap it and refresh its TTL.' Below this is a text input field and a blue button labeled 'Rewrap token'.

- **Random:** Generates cryptographically secure random data to create unique IDs, tokens, passwords and other data that requires a higher grade of randomness and security.



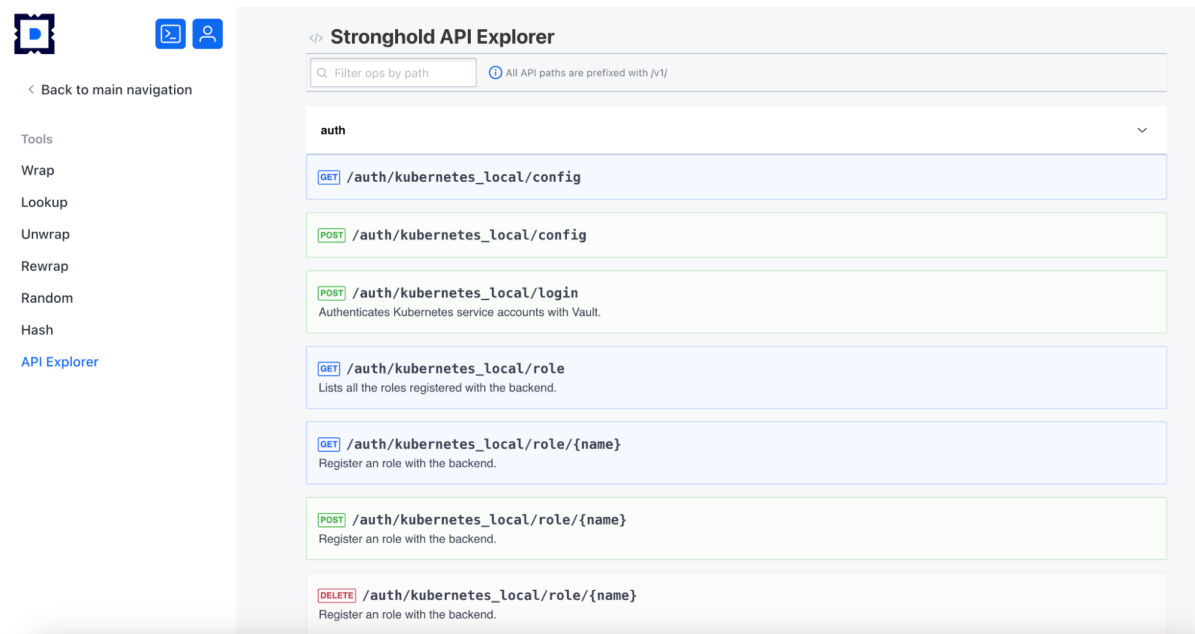
The screenshot shows the Deckhouse web interface. The left sidebar is identical to the previous ones, with 'Random' highlighted in blue. The main content area is titled 'Random Bytes'. It contains two input fields: 'Number of bytes' with the value '45673' and 'Output format' with the value 'hex'. Below these fields is a blue button labeled 'Generate'.

- **Hash:** Generates hashes for different data. Supports multiple caching algorithms.



The screenshot shows the 'Hash Data' tool. On the left is a navigation menu with options: Tools, Wrap, Lookup, Unwrap, Rewrap, Random, Hash (selected), and API Explorer. The main area has a title 'Hash Data'. Below it is an 'Input' text box. To the right of the input box is a button labeled 'Encode to base64'. Below the input box are two dropdown menus: 'Algorithm' (set to 'sha2-256') and 'Output format' (set to 'hex'). At the bottom is a blue 'Hash' button.

- **API Explorer:** Used for interacting with stronghold API via GUI.

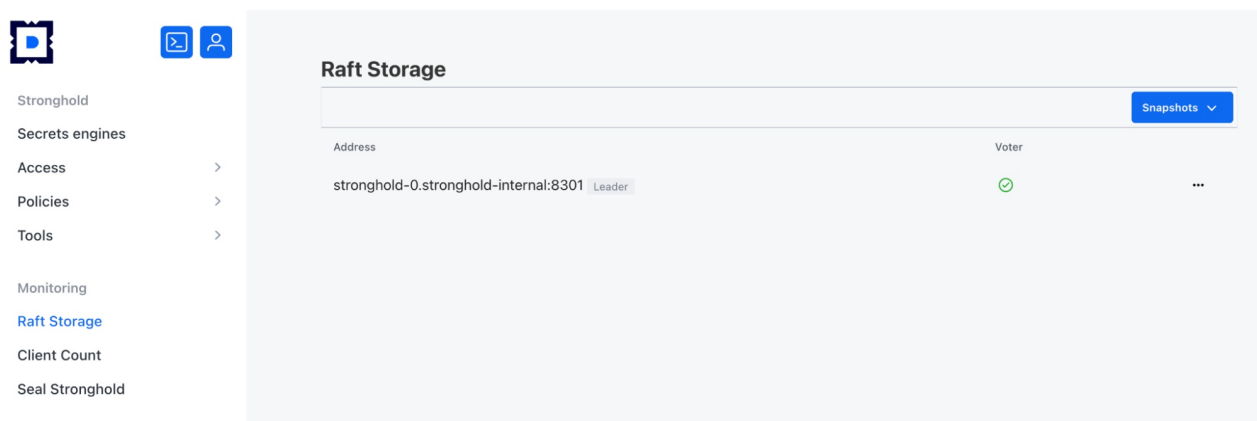


The screenshot shows the 'Stronghold API Explorer'. The left navigation menu is the same as the previous screenshot, with 'API Explorer' selected. The main area has a title 'Stronghold API Explorer'. Below the title is a search bar 'Filter ops by path' and a note 'All API paths are prefixed with /v1/'. A dropdown menu is set to 'auth'. Below it are several API endpoints listed with their methods and descriptions:
 

- GET** /auth/kubernetes\_local/config
- POST** /auth/kubernetes\_local/config
- POST** /auth/kubernetes\_local/login: Authenticates Kubernetes service accounts with Vault.
- GET** /auth/kubernetes\_local/role: Lists all the roles registered with the backend.
- GET** /auth/kubernetes\_local/role/{name}: Register an role with the backend.
- POST** /auth/kubernetes\_local/role/{name}: Register an role with the backend.
- DELETE** /auth/kubernetes\_local/role/{name}: Register an role with the backend.

## Monitoring stronghold Raft cluster

Use the “Raft Storage” section to monitor the stronghold Raft cluster. The left menu contains a navigation menu; the central pane shows the cluster leader, nodes, and a **Snapshots** button for backing up or restoring Raft data from a snapshot.



The screenshot shows the 'Raft Storage' section. The left navigation menu has 'Monitoring' selected, with 'Raft Storage' highlighted. The main area has a title 'Raft Storage'. Below the title is a 'Snapshots' button. Below that is a table with columns 'Address', 'Voter', and an ellipsis menu. The table contains one entry:
 

| Address                               | Voter  |     |
|---------------------------------------|--------|-----|
| stronghold-0.stronghold-internal:8301 | Leader | ... |

## Monitoring activity and load

To monitor activity and assess stronghold workload, go to the “Client Count” section. The left pane contains a navigation menu.

The central pane contains the following two tabs:

- **Dashboard** with a number of unique clients for the current month and buttons to change the period.

**Stronghold Usage Metrics**

Dashboard Configuration

This dashboard will surface Stronghold client usage over time. Clients represent a user or service that has authenticated to Stronghold. Documentation is available [here](#). Date queries are sent in UTC.

Client counting start date

Month Year Save

This date is when client counting starts. Without this starting point, the data shown is not reliable.

**No start date found**

In order to get the most from this data, please enter a start month above. Stronghold will calculate new clients starting from that month.

- **Configuration** that allows editing of metric collection settings (use the **Edit configuration** button for that).

**Stronghold Usage Metrics**

Dashboard Configuration

Edit configuration

**Usage data collection**

Enable or disable collecting data to track clients. On

**Retention period**

The number of months of activity logs to maintain for client tracking. 24

## Sealing and unsealing the secrets storage

The “Seal Stronghold” section provides means to seal or unseal the secrets storage. The left pane contains a navigation menu. The central pane contains a button for sealing or unsealing the secret storage (depending on its current state).

When the storage is sealed, stronghold can’t read or write secrets.

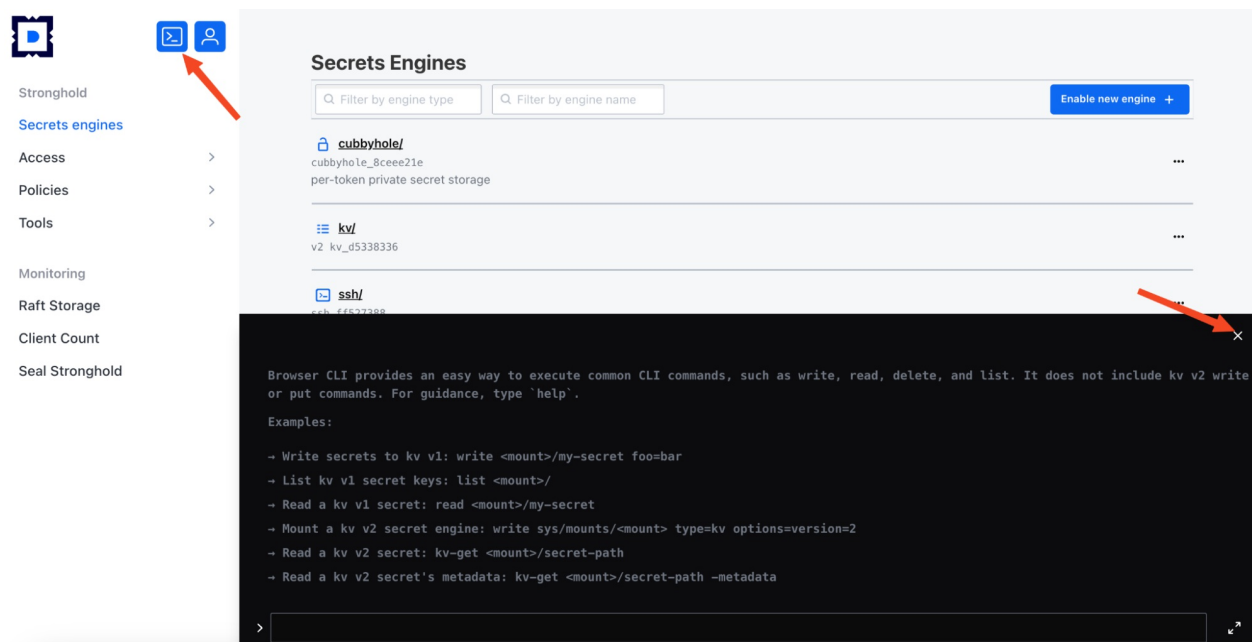
**Seal this Stronghold**

Sealing a Stronghold tells the Stronghold server to stop responding to any access operations until it is unsealed again. A sealed Stronghold throws away its root key to unlock the data, so it physically is blocked from responding to operations again until the Stronghold is unsealed again with the "unseal" command or via the API.

Seal

## Using the stronghold CLI

The stronghold CLI tool can perform operations related to secrets, policies, users, and more. You can launch it from any interface section via the CLI button in the upper-left corner. Click again to close it.



## Additional resources

- Module `stronghold` documentation

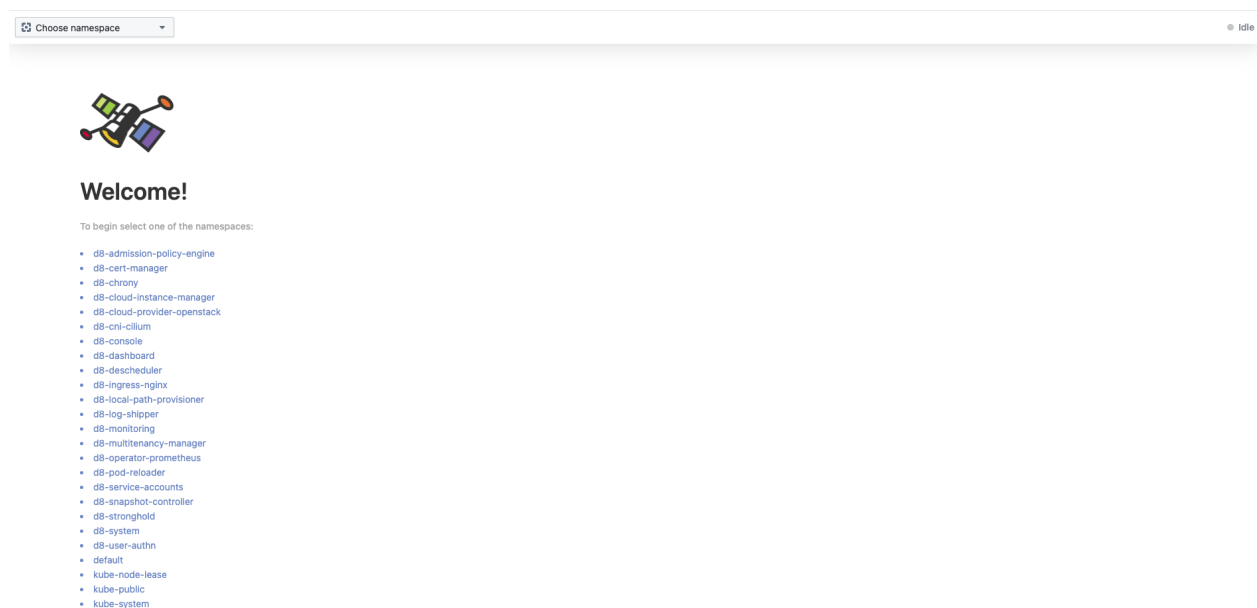
## Web interface for visualizing the cluster network stack

The web interface for cluster network stack visualization is intended for operational diagnostics and analysis of network interactions. This interface enables tracking of communications between pods, services, and external resources, analyzing network activity and pinpointing network issues. The interface is implemented by the `cilium-hubble` module.

## Accessing the web interface

To access the web interface for visualizing the cluster network stack, open the URL `hubble.<CLUSTER_NAME_TEMPLATE>`, where `<CLUSTER_NAME_TEMPLATE>` is the string matching the cluster's DNS name template, as specified by the global parameter `modules.publicDomainTemplate`.

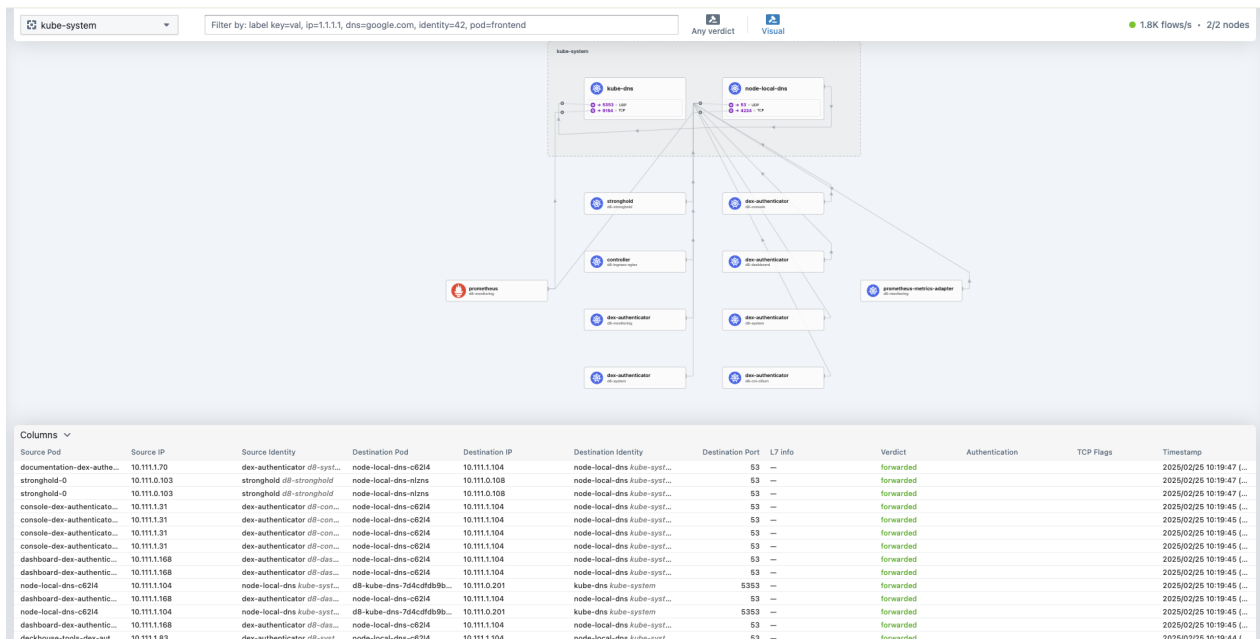
When you navigate to `hubble.<CLUSTER_NAME_TEMPLATE>`, you will see a namespace selection screen to choose which namespace's network stack will be displayed.



Select a namespace using the dropdown in the top-left corner or by clicking the desired namespace name in the list at the center of the screen.

After selecting a namespace, you will see the network stack visualization and analysis tools. The interface consists of the following parts:

- Top panel with filters and a brief cluster summary (number of flows and number of nodes). For more details on filtering what is displayed, see the section [Filtering data for display](#).
- Network flow diagram. For more information on working with the visual flow diagram, see [Working with the network flow diagram](#).
- Network flows and events table. For details on using the table, see [Working with the network flows and events table](#).

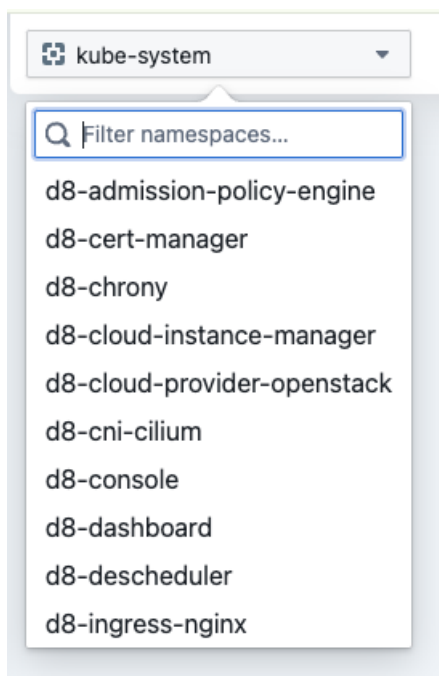


Data in the diagram and the network flows table is displayed in real time.

## Filtering data for display

To filter the displayed information about the network stack and flows, use the top panel with filters. This panel contains:

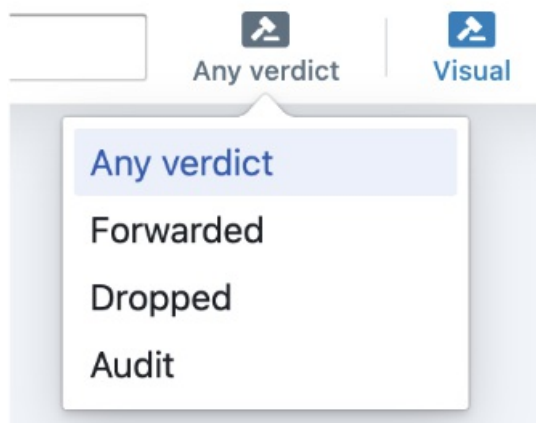
- Namespace selector (dropdown on the left side of the panel).



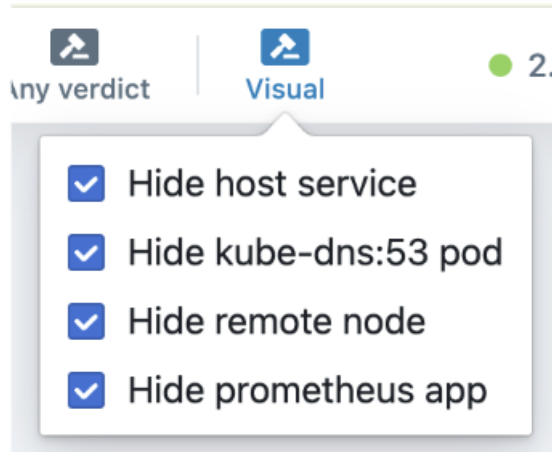
- Resource selector (input field in the center of the panel) to choose which namespace resources' flows to display.

Filter by: label key=val, ip=1.1.1.1, dns=google.com, identity=42, pod=frontend

- Flow verdict selector, based on the decision ("verdict") made by Cilium for each flow.



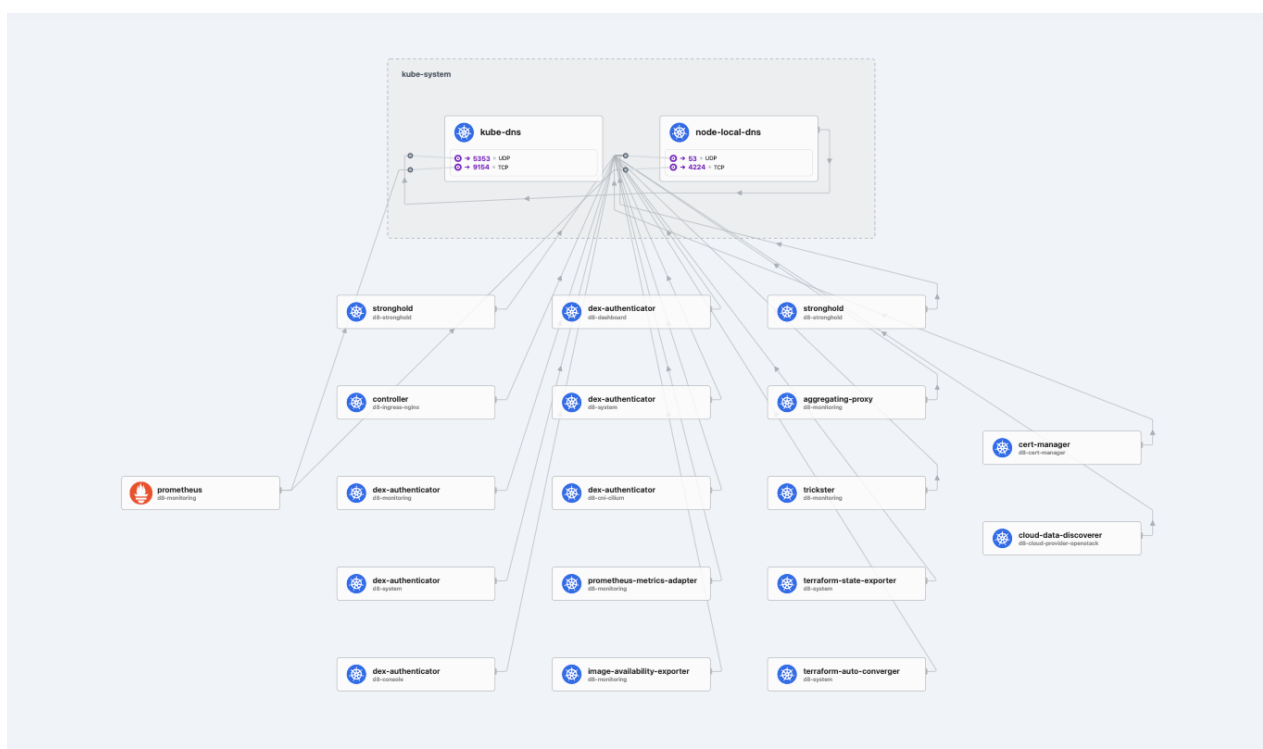
- Diagram element selector for choosing which elements in the namespace to show on the diagram.



## Working with the network flow diagram

The network flow diagram for the selected namespace appears in the middle of the screen, showing the network stack visualization and analysis tools. The diagram displays the resources of the chosen namespace inside a rectangle labeled with that namespace's name, as well as external elements they interact with.

To view detailed information (labels, network interactions, etc.) for a specific resource on the diagram, click on that resource.





## Working with the network flows and events table

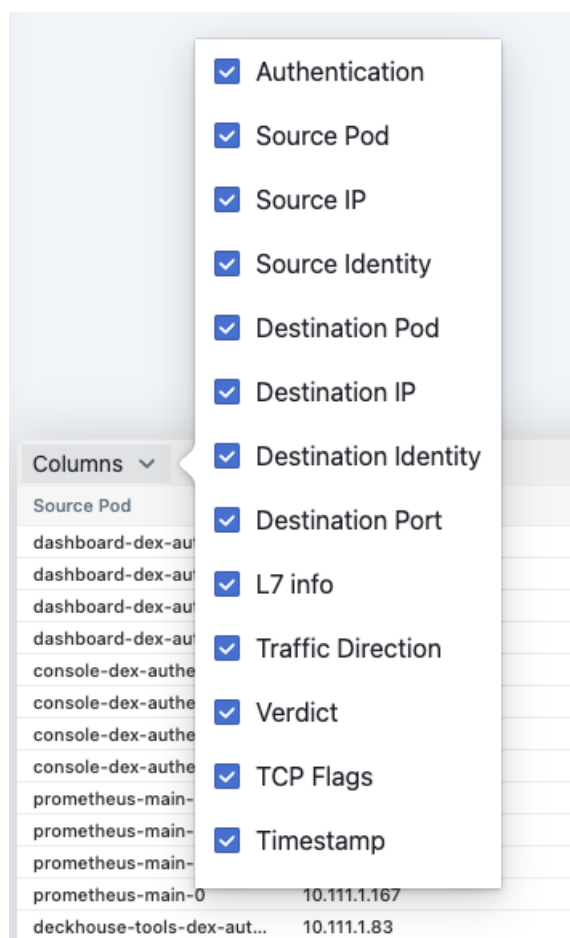
Each row in the table contains the following information about a network flow:

- Source Pod (pod name of the flow source)
- Source IP (IP address of the source pod)
- Source Identity (entity identifier for the source)
- Destination Pod (pod name of the flow destination)
- Destination IP (IP address of the destination pod)
- Destination Identity (entity identifier for the destination)
- Destination Port (destination port number)
- L7 Info (application-layer information, if the flow uses HTTP)
- Verdict (the decision made by Cilium for the flow)
- Authentication (authentication results for the flow, if applicable)
- TCP Flags (TCP flags associated with the flow)
- Timestamp (time when the flow occurred)

| Columns ▾                  |              |                             |                          |                |                             |                  |         |           |                |           |                           |
|----------------------------|--------------|-----------------------------|--------------------------|----------------|-----------------------------|------------------|---------|-----------|----------------|-----------|---------------------------|
| Source Pod                 | Source IP    | Source Identity             | Destination Pod          | Destination IP | Destination Identity        | Destination Port | L7 Info | Verdict   | Authentication | TCP Flags | Timestamp                 |
| dashboard-dex-authentic... | 10.111.1.168 | dex-authenticator d8-das... | node-local-dns-c6214     | 10.111.1.104   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:40 (...) |
| dashboard-dex-authentic... | 10.111.1.168 | dex-authenticator d8-das... | node-local-dns-c6214     | 10.111.1.104   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:40 (...) |
| node-local-dns-c6214       | 10.111.1.104 | node-local-dns kube-syst... | d8-kube-dns-764cd8b9b... | 10.111.0.215   | kube-dns kube-system        | 5353             | —       | forwarded |                |           | 2025/02/25 12:11:40 (...) |
| node-local-dns-c6214       | 10.111.1.104 | node-local-dns kube-syst... | d8-kube-dns-764cd8b9b... | 10.111.0.215   | kube-dns kube-system        | 5353             | —       | forwarded |                |           | 2025/02/25 12:11:40 (...) |
| dashboard-dex-authentic... | 10.111.1.168 | dex-authenticator d8-das... | node-local-dns-c6214     | 10.111.1.104   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:40 (...) |
| dashboard-dex-authentic... | 10.111.1.168 | dex-authenticator d8-das... | node-local-dns-c6214     | 10.111.1.104   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:40 (...) |
| console-dex-authenticat... | 10.111.1.31  | dex-authenticator d8-con... | node-local-dns-c6214     | 10.111.1.104   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:40 (...) |
| console-dex-authenticat... | 10.111.1.31  | dex-authenticator d8-con... | node-local-dns-c6214     | 10.111.1.104   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:40 (...) |
| console-dex-authenticat... | 10.111.1.31  | dex-authenticator d8-con... | node-local-dns-c6214     | 10.111.1.104   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:40 (...) |
| console-dex-authenticat... | 10.111.1.31  | dex-authenticator d8-con... | node-local-dns-c6214     | 10.111.1.104   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:40 (...) |
| prometheus-metrics-adap... | 10.111.0.133 | prometheus-metrics-adap...  | node-local-dns-nlzm...   | 10.111.0.108   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:39 (...) |
| prometheus-metrics-adap... | 10.111.0.133 | prometheus-metrics-adap...  | node-local-dns-nlzm...   | 10.111.0.108   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:39 (...) |
| prometheus-metrics-adap... | 10.111.0.133 | prometheus-metrics-adap...  | node-local-dns-nlzm...   | 10.111.0.108   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:39 (...) |
| prometheus-metrics-adap... | 10.111.0.133 | prometheus-metrics-adap...  | node-local-dns-nlzm...   | 10.111.0.108   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:39 (...) |
| prometheus-metrics-adap... | 10.111.0.133 | prometheus-metrics-adap...  | node-local-dns-nlzm...   | 10.111.0.108   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:39 (...) |
| prometheus-metrics-adap... | 10.111.0.133 | prometheus-metrics-adap...  | node-local-dns-nlzm...   | 10.111.0.108   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:39 (...) |
| prometheus-metrics-adap... | 10.111.0.133 | prometheus-metrics-adap...  | node-local-dns-nlzm...   | 10.111.0.108   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:39 (...) |
| prometheus-metrics-adap... | 10.111.0.133 | prometheus-metrics-adap...  | node-local-dns-nlzm...   | 10.111.0.108   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:39 (...) |
| prometheus-metrics-adap... | 10.111.0.133 | prometheus-metrics-adap...  | node-local-dns-nlzm...   | 10.111.0.108   | node-local-dns kube-syst... | 53               | —       | forwarded |                |           | 2025/02/25 12:11:39 (...) |

## Configuring the column set

To configure which columns are shown in the table, click the Columns button in the top-left corner of the table and select the columns you need.



## Viewing row details

To view detailed information for a specific row in text form, click anywhere within that row. The information will appear on the right side of the table. These details are shown regardless of which columns are currently selected for display.

| Flow Details             |                                                                                                                                                                                                                                |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Timestamp</b>         | 2025-02-25T09:18:37.673Z                                                                                                                                                                                                       |
| <b>Verdict</b>           | <b>forwarded</b>                                                                                                                                                                                                               |
| <b>Traffic direction</b> | ingress                                                                                                                                                                                                                        |
| <b>Cilium event type</b> | to-endpoint                                                                                                                                                                                                                    |
| <b>Source pod</b>        | <a href="#">stronghold-0</a>                                                                                                                                                                                                   |
| <b>Source identity</b>   | 15378                                                                                                                                                                                                                          |
| <b>Source labels</b>     | <a href="#">instance=stronghold</a><br><a href="#">name=stronghold</a><br><a href="#">apps.kubernetes.io/pod-index=0</a><br><a href="#">component=server</a><br><a href="#">io.cilium.k8s.namespace.labels.managed-by=Helm</a> |

## Using 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.

## Creating a project

1. To create a project, create a Project custom resource and specify the project template name in the `.spec.projectTemplateName` field.
2. In the `.spec.parameters` field, specify parameter values for the

2. in the `.spec.parameters` field, specify parameter values for the `.spec.parametersSchema.openAPIV3Schema` section of the `ProjectTemplate` custom resource.

Example of creating a project using Project 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 project status, run the following command:

```
d8 k get projects my-project
```

A successfully created project will have a `Deployed` status. If the `Error` status is displayed instead, run the same command with the `-o yaml` flag to get details about the cause:

```
d8 k get projects my-project -o yaml
```

## Automatically creating a project from a namespace

You can create a new project from a namespace by adding the `projects.deckhouse.io/adopt` annotation to it. For example:

1. Create a new namespace:

```
d8 k create ns test
```

2. Annotate it:

```
d8 k annotate ns test projects.deckhouse.io/adopt=""
```

3. Check that the project was created:

```
d8 k get projects
```

In the output list of projects, you should see the newly created project corresponding to the namespace:

| 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 an existing one.



Note that changing the template might cause resource conflicts. If the new template's chart defines resources that already exist in the namespace, the template can't be applied.

For details on project templates and their creation, refer to the [Administration](#) section.

## Additional resources

- Module `multitenancy-manager` custom resources
- Module `secret-copier` documentation

## Application and Infrastructure Monitoring

This section is intended for project users of the Deckhouse Kubernetes Platform (DKP).

DKP includes a built-in monitoring system that provides convenient tools for observing the state of the infrastructure and applications.

By default, DKP provides a predefined set of dashboards and alerts that help track key application health metrics. They are available through the “Monitoring” section of the Deckhouse web interface.

In addition, users can:

- Collect metrics from their own applications.
- Create custom dashboards to visualize required metrics.
- Configure custom alerts and override alert thresholds.

For more information about advanced monitoring capabilities, including dashboard and metrics management, refer to the documentation for the `observability` module.

## Available by default

After installing DKP, users have access to a basic set of tools for monitoring the cluster out of the box.

## Dashboards

Dashboards display charts with data on CPU usage, memory consumption, disk activity, and network traffic, broken down by pods, nodes, or namespaces.

In the “Monitoring” → “Dashboards” section of the Deckhouse web interface, users can access the following dashboard groups:

- **Ingress Nginx:** Metrics related to the Ingress NGINX Controller operation, including virtual host status, response codes, and request latency.
- **Resource consumption (Main):** Key cluster and application metrics, including resource utilization and status of pods, controllers, and namespaces.
- **Security:** Metrics related to cluster security.

## Alerts

Alerts are automated notifications that report events requiring attention, such as metric threshold violations or component availability issues. For most alerts, trigger thresholds can be overridden if necessary.

By default, a DKP cluster includes alerts for the following types of events:

- Certificate expiration, as well as errors during certificate issuing or renewal ( `cert-manager`, `extended-monitoring`, and `ingress-nginx` modules).

- Container image pull failures, including authentication and authorization issues, invalid image names, missing images in a registry, or registry unavailability ( `extended-monitoring` module).
- Workload execution errors for resources such as CronJob, Deployment, DaemonSet, and StatefulSet, including pod creation failures, unavailable replicas, and scheduling errors ( `extended-monitoring` module).
- Metric exporter unavailability, preventing Prometheus from scraping metrics ( `extended-monitoring` module).
- Disk space issues, including insufficient storage space or inode exhaustion on PVCs ( `extended-monitoring` module).
- Ingress NGINX Controller errors, including a high rate of `5xx` responses from backends ( `extended-monitoring` module).
- Network performance issues ( `monitoring-ping` module).

## Monitoring configuration

The following monitoring configuration options are available to DKP users:

- **Monitoring user applications:** You can configure metrics collection from your application by following the instructions.
- **Creating custom dashboards:** You can add specialized dashboards using the `GrafanaDashboardDefinition` resource.
- **Configuring custom alerts:** You can define new alerting rules in one of two ways:
  - **Recommended:** Using the `ClusterObservabilityMetricsRulesGroup` and `ObservabilityMetricsRulesGroup` resources of the `observability` module.
  - **Outdated:** Using the `CustomPrometheusRules` resource of the `prometheus` module. To see an example, refer to the module documentation.

## Additional resources

- Module `console` documentation
- Module `observability` documentation
- Module `prometheus` documentation

## Monitoring dashboards

In this section, you will learn how to work with dashboards to analyze the state of the Deckhouse Kubernetes Platform (DKP) and the applications running in it.

Dashboards are collections of charts and tables that display application performance data. They provide information about CPU usage, memory consumption, disk and network activity, as well as the status of Pods, controllers, nodes, and namespaces.

## Dashboard types

DKP features preinstalled and custom dashboards, which can be created in several ways.

| Dashboard type                                                          | Description                                                                                                                                                                                                                                                     |
|-------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Preinstalled                                                            | Ready-to-use dashboards that are already installed in DKP. Designed to monitor the state of running applications.                                                                                                                                               |
| Custom dashboards created using the <code>observability</code> module   | Custom dashboards created using the <code>ObservabilityDashboard</code> resource at the namespace level, with support for access control.<br><br>This is the recommended way to work with dashboards.                                                           |
| Custom dashboards created using <code>GrafanaDashboardDefinition</code> | Custom dashboards created using the <code>GrafanaDashboardDefinition</code> resource at the cluster level. They require elevated privileges and do not support access control.<br><br>This is a legacy approach that will be deprecated in future DKP versions. |

## Preinstalled dashboards

DKP users have access to a basic set of dashboards for monitoring the state of running applications. Dashboards are available in the Deckhouse web UI under “Monitoring” → “Dashboards”.

**i** Preinstalled dashboards are not available for editing.

## Ingress Nginx

Dashboards for monitoring the operation of the Ingress controller. They include metrics reflecting the state of virtual hosts, HTTP response statistics, and request processing latency data.

Available dashboards:

- **Namespaces:** Aggregated Ingress resource metrics by namespace.
- **Namespace Detail:** Detailed information about Ingress resources in a selected namespace.
- **VHosts:** An overview of the state of virtual hosts.
- **VHost Detail:** Detailed information about a selected virtual host.

## Resource consumption (Main)

A set of dashboards for analyzing application resource consumption. They are intended for load assessment, resource issue detection, and workload health analysis.

Available dashboards:

- **Namespaces:** Summary information for all namespaces.
- **Namespace:** Key resource usage metrics for a selected namespace.
- **Namespace / Controller:** Resource usage statistics for controllers within a selected namespace.
- **Namespace / Controller / Pod:** Detailed metrics for individual Pods.

## Security

Dashboards that contain metrics related to the security of cluster components.

Available dashboards:

- **Admission policy engine** — metrics of the `admission-policy-engine` module, including information about policy checks and enforcement.

## Custom dashboards

DKP users can create custom dashboards in several ways, depending on the requirements for access control and the dashboard scope.

## Using the observability module

The `observability` module extends the functionality of the `prometheus` module and the Deckhouse web UI by providing additional capabilities for flexible metric visualization and access control.

The module introduces new dashboard types, including namespace-scoped resources. This allows users to create and manage their own dashboards without requiring permissions for cluster-level objects. The module also simplifies dashboard editing by allowing users to configure dashboards directly in the web UI, without manually managing resources.

**i** Before using these resources, make sure that the `observability` module is enabled in the cluster. If necessary, contact your DKP administrator.

The following resources are available for creating dashboards:

- **ObservabilityDashboard:** Dashboards scoped to a namespace. They are displayed in the Deckhouse web UI under “Monitoring” → “Projects”.

Example:

```

apiVersion: observability.deckhouse.io/v1alpha1
kind: ObservabilityDashboard
metadata:
  name: example-dashboard
  namespace: my-namespace
  annotations:
    metadata.deckhouse.io/category: "Apps"
    metadata.deckhouse.io/title: "Example dashboard"
spec:
  definition: |
    {
      "title": "Example dashboard",
      ...
    }

```

- **ClusterObservabilityDashboard**: Dashboards for visualizing cluster components. They are displayed in the Deckhouse web UI under “Monitoring” → “System”.

Example:

```

apiVersion: observability.deckhouse.io/v1alpha1
kind: ClusterObservabilityDashboard
metadata:
  name: example-dashboard
  annotations:
    metadata.deckhouse.io/category: "Apps"
    metadata.deckhouse.io/title: "Example dashboard"
spec:
  definition: |
    {
      "title": "Example dashboard",
      ...
    }

```

- **ClusterObservabilityPropagatedDashboard**: Dashboards that extend the list of dashboards from the two categories above. Such dashboards are automatically added to the Deckhouse web UI and are displayed under “Monitoring” → “System and Monitoring” → “Projects”. They become available to users who have permissions for the corresponding namespace or system section.

Example:

```

apiVersion: observability.deckhouse.io/v1alpha1
kind: ClusterObservabilityPropagatedDashboard
metadata:
  name: example-dashboard
  annotations:
    metadata.deckhouse.io/category: "Apps"
    metadata.deckhouse.io/title: "Example dashboard"
spec:
  definition: |
    {
      "title": "Example dashboard",
      ...
    }

```

## Access control

Access to dashboards is configured using the mechanisms of the current role-based access control (RBAC) model.

Depending on the dashboard type (system or user), permissions are granted for the following resources:

- `observabilitydashboards.observability.deckhouse.io`: Namespace-scoped dashboards.
- `clusterobservabilitydashboards.observability.deckhouse.io`: System dashboards.
- `clusterobservabilitypropagateddashboards.observability.deckhouse.io`: Dashboards propagated to all users.

The following permissions are required to perform operations on dashboards:

- reading: `get`
- creating and editing: `create`, `update`, `patch`, `delete`

Access to metrics used in dashboards is also controlled via RBAC. Depending on the granted permissions, metric filtering is performed automatically.

The following access scenarios are supported:

- Namespace users can access only the metrics of their own namespace. RBAC access to the `metrics.observability.deckhouse.io` resource is checked.
- DKP administrators have access to all system metrics:
  - Deckhouse metrics (`d8-*`)
  - Kubernetes metrics (`kube-*`)
  - metrics without the `namespace` label

RBAC access to the `clustermetrics.observability.deckhouse.io` resource is used.

- Metrics from user namespaces can also be available to administrators if they have the appropriate permissions for the `metrics.observability.deckhouse.io` resource.

Example configuration of ClusterRole and RoleBinding resources for read and edit access to metrics and dashboards:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: observability-editor
rules:
  - apiGroups: ["observability.deckhouse.io"]
    resources: ["metrics", "observabilitydashboards"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["observability.deckhouse.io"]
    resources: ["observabilitydashboards"]
    verbs: ["create", "update", "delete"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bind-observability-editor
  namespace: my-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: observability-editor
subjects:
  - kind: User
    name: user@example.com
    apiGroup: rbac.authorization.k8s.io
```

## Converting dashboards from GrafanaDashboardDefinition

To migrate dashboards created using the legacy `GrafanaDashboardDefinition` resource to one of the formats supported by the `observability` module, edit each dashboard manifest manually. Pay attention to the following key differences:

| GrafanaDashboardDefinition format                                      | observability module format                                                                    |
|------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| The Grafana folder is specified in the <code>spec.folder</code> field. | The folder is specified using the <code>observability.deckhouse.io/category</code> annotation. |



| GrafanaDashboardDefinition format                                                      | observability module format                                                                                                                                                           |
|----------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The dashboard title is specified in the <code>title</code> field of the JSON manifest. | The title is specified using the <code>observability.deckhouse.io/title</code> annotation. If the annotation is missing, the <code>title</code> field from the JSON manifest is used. |

Conversion example:

- Old format:

```
apiVersion: deckhouse.io/v1
kind: GrafanaDashboardDefinition
metadata:
  name: example-dashboard
spec:
  folder: "Apps"
  json: '{
    "title": "Example Dashboard",
    ...
  }'
```

- New format:

```
apiVersion: observability.deckhouse.io/v1alpha1
kind: ObservabilityDashboard
metadata:
  name: example-dashboard
  namespace: my-namespace
  annotations:
    metadata.deckhouse.io/category: "Apps"
    metadata.deckhouse.io/title: "Example Dashboard"
spec:
  definition: |
    {
      "title": "Example Dashboard",
      ...
    }
```

## Using GrafanaDashboardDefinition

**i** This is a legacy approach and is not recommended for new dashboards. Support for this method will be removed in future DKP versions.

To add a dashboard directly to Grafana, use the `GrafanaDashboardDefinition` resource.

Example:

```

apiVersion: deckhouse.io/v1
kind: GrafanaDashboardDefinition
metadata:
  name: my-dashboard
spec:
  folder: My folder # The folder in Grafana where the dashboard will be displayed.
  definition: |
    {
      "annotations": {
        "list": [
          {
            "builtIn": 1,
            "datasource": "-- Grafana --",
            "enable": true,
            "hide": true,
            "iconColor": "rgba(0, 211, 255, 1)",
            "limit": 100,
            ...

```

When using this method, consider the following limitations:

- Dashboards added via `GrafanaDashboardDefinition` cannot be modified through the Grafana UI.
- Alerts configured in the Dashboard panel do not work with datasource templates — such dashboards are considered invalid and are not imported. Starting with Grafana 9.0, legacy alerting has been deprecated and replaced with Grafana Alerting. Therefore, using legacy alerting (dashboard panel alerts) in dashboards is not recommended.
- If the dashboard does not appear in Grafana after applying the resource, the dashboard JSON file may contain an error. To view the logs of the component responsible for dashboard provisioning, use the following command:

```
d8 k logs -n d8-monitoring deployments/grafana-v10 dashboard-provisioner
```

## Additional resources

- Module `admission-policy-engine` documentation
- Module `console` documentation
- Module `observability` documentation
- Module `prometheus` custom resources

## Configuring Application Monitoring

Deckhouse Kubernetes Platform (DKP) supports four ways to connect an application to the monitoring system:

| Connection method                  | Description                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Via labels and annotations         | The simplest and fastest method, requiring only metadata to be added to a Service or Pod. Allows you to configure basic monitoring parameters.                                                                                                                                                                                                                     |
| Using PodMonitor or ServiceMonitor | An advanced monitoring configuration method for cases where Prometheus relabeling rules are required. Provides flexible control over metrics collection and label processing and lets you add custom labels into metrics. This approach is suitable for complex monitoring scenarios but requires a deeper understanding of Prometheus and its scraping mechanism. |
| Using ScrapeConfig                 | A monitoring configuration method that is as close as possible to the native Prometheus configuration structure. Provides full control over scrape settings, including relabeling, and allows collecting metrics both from Kubernetes and from targets located outside the cluster.                                                                                |

| Connection method                               | Description                                                                                                                                              |
|-------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Availability monitoring using blackbox-exporter | A method for monitoring endpoint availability using probes. It is integrated using blackbox-exporter, which must be installed in the cluster separately. |

## Configuring metrics collection via labels and annotations

**i** This section describes a basic application integration scenario. For advanced configuration options, refer to additional annotations.

1. Make sure the `monitoring-custom` module is enabled. If necessary, contact your DKP administrator.
2. Ensure that the application exposing metrics does so in the Prometheus format.
3. Add the `prometheus.deckhouse.io/custom-target` label to the Service or Pod that should be connected to monitoring. The label value defines the name in the Prometheus targets list.

Example:

```
labels:
  prometheus.deckhouse.io/custom-target: my-app
```

It is recommended to use the application name as the value of the `prometheus.deckhouse.io/custom-target` label, which allows it to be uniquely identified in the cluster.

The label format must comply with Kubernetes requirements: no more than 63 characters, which can include alphanumeric characters ( `[a-z0-9A-Z]` ), as well as hyphens ( `-` ), underscores ( `_` ), dots ( `.` ).

If the application is deployed in the cluster more than once (staging, testing, etc.) or even deployed several times in one namespace, one common name is sufficient, since all metrics will have `namespace`, `pod` labels anyway, and if access is through Service, the `service` label. This is the name that uniquely identifies the application in the cluster, not its single installation.

4. Specify the name `http-metrics` and `https-metrics` for the port from which metrics need to be collected for HTTP or HTTPS connection respectively.

If this is not possible (for example, the port is already defined and named differently), use the following annotations:

- `prometheus.deckhouse.io/port: port_number`: To specify the port.
- `prometheus.deckhouse.io/tls: "true"`: If metrics collection will be over HTTPS.

**i** When specifying an annotation on a Service, you must use `targetPort` as the port value. That is, the port that is open and listened to by the application, not the Service port.

- Example 1:

```
ports:
- name: https-metrics
  containerPort: 443
```

- Example 2:

```
annotations:
  prometheus.deckhouse.io/port: "443"
  prometheus.deckhouse.io/tls: "true" # If metrics are provided over HTTP,
do not specify this annotation.
```

5. When using service mesh Istio in STRICT mTLS mode, specify the following annotation for metrics collection on Service or Pod: `prometheus.deckhouse.io/istio-mtls: "true"`. It is important

that application metrics should be exported over HTTP protocol without TLS.

Example:

```
annotations:
  prometheus.deckhouse.io/istio-mtls: "true"
```

## Example of configuring metrics collection from a Service

Below is an example of setting up metrics collection from a 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, the service
port with the name http-metrics or https-metrics will be used.
    prometheus.deckhouse.io/path: "/my_app/metrics" # Set to /metrics by
default.
    prometheus.deckhouse.io/query-param-format: "prometheus" # Set to '' by default.
    prometheus.deckhouse.io/allow-unready-pod: "true" # By default, Pods NOT in
Ready state are ignored.
    prometheus.deckhouse.io/sample-limit: "5000" # By default, no more than
5000 metrics are accepted from a single Pod.
spec:
  ports:
    - name: my-app
      port: 8060
    - name: http-metrics
      port: 8061
      targetPort: 8061
  selector:
    app: my-app
```

## Example of configuring metrics collection from a Deployment

Below is an example of setting up metrics collection from a 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, no more than 5000
metrics are accepted from a single Pod.
    spec:
      containers:
        - name: my-app
          image: my-app:1.7.9
          ports:
            - name: https-metrics
              containerPort: 443

```

## Additional annotations for advanced configuration

For more precise application monitoring configuration, you can specify additional annotations for the Pod or Service for which monitoring is configured:

- `prometheus.deckhouse.io/path`: Path for metrics collection (default: `/metrics`).
- `prometheus.deckhouse.io/query-param-$name`: GET parameters that will be converted to a map of the form `$name=$value` (default: `''`). You can specify several such annotations. For example, `prometheus.deckhouse.io/query-param-foo=bar` and `prometheus.deckhouse.io/query-param-bar=zxc` will be converted to a request like `http://...?foo=bar&bar=zxc`.
- `prometheus.deckhouse.io/allow-unready-pod`: Allows metrics collection from pods in any state (by default, metrics are collected only from pods in `Ready` state). This option is useful in rare cases. For example, if your application starts very slowly (data is loaded into the database or caches are warmed up at startup), but useful metrics are already provided during startup that help monitor the application startup.
- `prometheus.deckhouse.io/sample-limit`: How many samples are allowed to be collected from a pod (5000 by default). The default value protects against situations where the application suddenly starts providing too many metrics, which can disrupt the entire monitoring system. The annotation must be placed on the same resource where the `prometheus.deckhouse.io/custom-target` label is attached.

## Configuring metrics collection using PodMonitor or ServiceMonitor resources

DKP supports connecting applications using two functionally similar resources:

- **PodMonitor** (recommended): Discovers Pods directly and collects metrics from their containers. In most cases, this is the preferred option, as it works directly with Pods and does not depend on the presence of Services.
- **ServiceMonitor**: Discovers Services and collects metrics from the Pods behind them. Services are used as a source of metadata (such as labels), while the actual metrics scraping is performed against the Pod addresses included in the corresponding endpoints.

Both resources let you configure the scrape interval, paths, TLS settings, relabeling rules, and other parameters.

The difference between these resources lies in the source of the collected metrics. Use PodMonitor if you need to scrape metrics directly from Pods, and ServiceMonitor if your application exposes metrics via a Service.

To connect an application to the monitoring system using one of these resources, follow these steps:

1. Add the `prometheus.deckhouse.io/monitor-watcher-enabled: "true"` label to the namespace where the PodMonitor or ServiceMonitor will be created:

```
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/monitor-watcher-enabled: "true"
```

2. Create a PodMonitor or ServiceMonitor resource, specifying the required `prometheus: main` label and the target endpoint parameters.

The `spec.selector.matchLabels` field defines the objects to use for metrics collection. For PodMonitor, these are target pod labels and for ServiceMonitor, these are target service labels.

PodMonitor example:

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: example-app
  namespace: frontend
  labels:
    prometheus: main
spec:
  selector:
    matchLabels:
      # Label of pods targeted for metrics collection.
      app: example-app
  podMetricsEndpoints:
    - port: metrics
      interval: 30s
      path: /metrics
```

ServiceMonitor example:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: example-app
  namespace: frontend
  labels:
    prometheus: main
spec:
  selector:
    matchLabels:
      # Label of a service that exposes application metrics.
      app: example-app
  endpoints:
    - port: web
      interval: 30s
      path: /metrics
```

If necessary, configure additional settings using the reference for available resource parameters: PodMonitor, ServiceMonitor.

## Adding custom labels to metrics

If you need to add additional labels directly to the collected metrics, use metric relabeling rules (via the `metricRelabelings` parameter).

The example below shows a PodMonitor resource that adds the label `application` with the value `dotnet` to the collected metrics:

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: example-app
  namespace: frontend
  labels:
    prometheus: main
spec:
  selector:
    matchLabels:
      app: example-app
  podMetricsEndpoints:
    - port: metrics
      interval: 30s
      path: /metrics
      metricRelabelings:
        - targetLabel: application
          replacement: dotnet
          action: Replace
```

After applying this rule, an additional label will appear in the metrics:

```
http_requests_total{
  application="dotnet",
  code="404",
  container="example-app",
  instance="10.111.5.69:8080",
  namespace="frontend",
  pod="example-app-75476c469d-rqq4s"
}
```

## Configuring metrics collection via `scrape_configs` using the `ScrapeConfig` resource

ScrapeConfig is a custom resource that lets you configure the `scrape_config` section of Prometheus configuration, providing full control over the metrics scraping process.

To connect an application to the monitoring system, follow these steps:

1. Add the `prometheus.deckhouse.io/scrape-configs-watcher-enabled: "true"` label to the namespace where the ScrapeConfig will be created:

```
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/scrape-configs-watcher-enabled: "true"
```

2. Create a ScrapeConfig resource with the required `prometheus: main` label:

```

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'

```

If necessary, configure additional settings using the reference for available resource parameters.

## Configuring metrics collection using blackbox-exporter

DKP supports availability metrics collection using blackbox-exporter, which is not included in DKP and must be installed separately in the cluster. The Probe custom resource is used to define availability checks (probes) executed by Prometheus.

To connect a Probe to the DKP monitoring system, follow these steps:

1. Add the `prometheus.deckhouse.io/probe-watcher-enabled: "true"` label to the namespace where the Probe will be created:

```

apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/probe-watcher-enabled: "true"

```

2. Create a Probe resource with the required `prometheus: main` label:



```

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

```

If necessary, configure additional settings using the [reference](#) for available resource parameters.

## Additional resources

- Module `monitoring-custom` [documentation](#)
- Module `operator-prometheus` [custom resources](#)

## Application delivery with Argo CD

This section describes how to organize application delivery with Argo CD in Deckhouse Kubernetes Platform (DKP).

Argo CD lets you describe applications declaratively and synchronize their state with the contents of a Git repository. The user specifies the manifest source, target cluster, namespace, and synchronization parameters, after which Argo CD deploys the application and keeps it in the target state.

In DKP, Argo CD instances are deployed with the `operator-argo` module. Typical work with Argo CD includes:

- creating or using an existing `AppProject` object;
- preparing the target namespace for the application;
- creating an `Application` object that describes the application source and synchronization rules;
- creating an application through the Argo CD web interface;
- creating an application with the `argocd` CLI utility.

## Prerequisites

Before you start, the following conditions must be met:

- the cluster administrator has enabled the `operator-argo` module;
- the administrator has deployed at least one Argo CD instance;
- the user has access to the required Argo CD instance and target namespaces.

If an Argo CD instance is not deployed yet, contact the administrator or follow the instructions in the [Running Argo CD](#) section.

## AppProject projects

`AppProject` is an Argo CD custom resource that defines the logical boundaries of a project. With it, you can define:

- which Git repositories are allowed as application sources;
- which clusters and namespaces applications may be deployed to;
- which cluster-wide and namespaced resources are allowed;
- which roles and access policies apply within the project.

Every Application object must reference a project through the `spec.project` parameter.

By default, Argo CD includes the `default` project. You can use it for first experiments and simple scenarios. For production environments, create separate AppProject objects to restrict access to repositories, clusters, and namespaces.

The manifest of the `default` AppProject object is shown below:

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: default
  namespace: argocd
spec:
  clusterResourceWhitelist:
    - group: '*'
      kind: '*'
  destinations:
    - namespace: '*'
      server: '*'
  sourceRepos:
    - '*'
```

## Preparing a namespace

Before deploying an application, create the target namespace and add the `argocd.argoproj.io/managed-by` label that indicates which Argo CD instance manages resources in this namespace.

Example:

```
apiVersion: v1
kind: Namespace
metadata:
  name: demo
  labels:
    argocd.argoproj.io/managed-by: argocd
```

In this example, the `demo` namespace will be managed by the Argo CD instance deployed in the `argocd` namespace.

## Deploying an application

You can create an application in Argo CD in several ways:

- declaratively — with an Application object;
- interactively — through the Argo CD web interface;
- with the `argocd` CLI utility.

## Creating an application with an Application object

An Application object is used to describe an application. It specifies:

- the Argo CD project (`spec.project`);
- the manifest or chart source (`spec.source`);
- the target cluster and namespace (`spec.destination`);
- the synchronization policy (`spec.syncPolicy`).

Example of an Application object:

```

apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: demo
  namespace: argocd
spec:
  destination:
    namespace: demo
    server: https://kubernetes.default.svc
  project: default
  source:
    path: helm-guestbook
    repoURL: https://github.com/argoproj/argocd-example-apps
    targetRevision: HEAD
  syncPolicy:
    # Enable automatic synchronization.
    automated:
      # Delete outdated resources.
      prune: true
      # Enable self-healing if third-party changes occur.
      selfHeal: true

```

After you create the Application object, Argo CD starts tracking the application state and synchronizing it with the repository contents. As a result, resources related to the `demo` application should appear in the `demo` namespace:

```

d8 k -n demo get deployment,svc,pod
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/demo-helm-guestbook  1/1      1              1            15s

```

| NAME                        | TYPE      | CLUSTER-IP    | EXTERNAL-IP | PORT(S) | AGE |
|-----------------------------|-----------|---------------|-------------|---------|-----|
| service/demo-helm-guestbook | ClusterIP | 10.222.177.84 | <none>      | 80/TCP  | 15s |

```

NAME                                READY    STATUS    RESTARTS    AGE
pod/demo-helm-guestbook-66d5d69ccd-xfkb7  1/1      Running    0            15s

```

## Creating an application through the Argo CD web interface

You can create an application through the Argo CD web interface. To do this:

1. Open the web interface of the required Argo CD instance.
2. Go to the “Applications” section.
3. Click “New App”.
4. Specify the application name, project, repository, revision, path to manifests or chart, and the target cluster and namespace.
5. If needed, configure automatic synchronization and additional parameters.
6. Click “Create”.

The form fields in the web interface correspond to the main parameters of the Application object: project, application source, target cluster, namespace, and synchronization policy.

## Creating an application with the `argocd` CLI utility

The `argocd` CLI utility lets you create and maintain applications from the command line. You can download the `argocd` binary from the “Documentation” section of the Argo CD web interface.

Before creating an application, authenticate with the following command:

```
argocd login <ARGOCD_DOMAIN>:443
```



When using SSO authentication, add the `--sso` flag to the login command.

Example of creating an application:

```
argocd app create guestbook \
  --repo https://github.com/argoproj/argocd-example-apps.git \
  --path guestbook \
  --dest-namespace demo \
  --dest-server https://kubernetes.default.svc \
  --directory-recurse \
  --sync-policy automated \
  --self-heal \
  --auto-prune
```

As with creating an Application object, the command specifies the application source, target cluster, namespace, and synchronization policy.

To view the status of a deployed application, use the `argocd app get <APP_NAME>` command, for example:

```
argocd app get guestbook
Name:          argocd/guestbook
Project:       default
Server:        https://kubernetes.default.svc
Namespace:     demo
URL:           https://argocd.192.168.0.235.sslip.io/applications/guestbook
Source:
- Repo:        https://github.com/argoproj/argocd-example-apps.git
  Target:
  Path:         guestbook
SyncWindow:    Sync Allowed
Sync Policy:   Automated (Prune)
Sync Status:   Synced to (8088f4c)
Health Status: Healthy
```

| GROUP | KIND                         | NAMESPACE | NAME         | STATUS | HEALTH  | HOOK | MESSAGE |
|-------|------------------------------|-----------|--------------|--------|---------|------|---------|
| apps  | Deployment                   | demo      | guestbook-ui | Synced | Healthy |      |         |
|       | deployment.apps/guestbook-ui |           | unchanged    |        |         |      |         |
|       | Service                      | demo      | guestbook-ui | Synced | Healthy |      |         |

## Additional resources

- Official Argo CD website
- Official Argo CD Operator website
- Module `operator-argo` documentation

## Marketplace

This section describes the ways to use Marketplace in the Deckhouse Kubernetes Platform (DKP).

Marketplace lets you install ready-made applications into your namespace from registries connected by the cluster administrator. Each application is installed as an Application resource and can exist in multiple independent instances — for example, separate Redis instances for caching and sessions in the same namespace.

**i** Marketplace are available starting from DKP version 1.76.

## Prerequisites

Before you can install an application, the cluster administrator must connect at least one package registry. Ask your administrator to check that `ApplicationPackageVersion` objects are available:

```
d8 k get apv
```

If the output is empty, the registry has not been scanned yet — contact your administrator.

The Installing and managing applications section describes how to browse available versions, install, update, and delete applications.

The Troubleshooting section covers how to diagnose and resolve problems with installed applications.

## Additional resources

- Global custom resources

# Installing and managing applications

## Browsing available package versions

To list all available package versions, use the following command (the short name `apv` can be used):

```
d8 k get apv
```

Example output:

| NAME<br>USED BY               | PACKAGE  | REPOSITORY  | TRANSITION TIME | METADATA LOADED |
|-------------------------------|----------|-------------|-----------------|-----------------|
| my-registry-redis-v7.2.0<br>1 | redis    | my-registry | 2d              | True            |
| my-registry-redis-v7.3.0      | redis    | my-registry | 5h              | True            |
| my-registry-pg-v15.0.0<br>2   | postgres | my-registry | 2d              | True            |

To filter by package name, use the following command (the example filters versions of the `redis` package):

```
d8 k get apv -l package=redis
```

To see which registries provide a specific package, use the following command (the short name `ap` can be used):

```
d8 k get ap redis \
-o jsonpath='{.status.availableRepositories}'
```

**i** Only versions with `MetadataLoaded=True` can be installed. This 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.

## Installing an application

To install an application, create an Application object in the desired namespace.

Example manifest for installing Redis from the `redis` package version `v7.2.0` with `maxmemory` settings:

```

apiVersion: deckhouse.io/v1alpha1
kind: Application
metadata:
  name: redis-cache
  namespace: my-app
spec:
  packageName: redis
  packageVersion: "v7.2.0"
  # Can be omitted if only one repository provides this package
  packageRepositoryName: my-registry
  settings:
    replicas: 3
    maxmemory: "256mb"

```

**i** `spec.settings` is validated against the OpenAPI schema defined in the package. If the schema rejects your settings, the Application will not be created. Check the package's `ApplicationPackageVersion` for schema details.

## Naming constraints

The Application name ( `metadata.name` ) must be **at most 24 characters**. This is required because all pods are prefixed with the instance name: a 24-char instance name + 24-char resource name + 15-char Deployment suffix fits within the Kubernetes pod name limit of 63 characters.

## Checking application status

To get a brief status of an application, use the following command (the short name `app` can be used):

```
d8 k get app -n <NAMESPACE> <APPLICATION_NAME>
```

Example output:

| NAME        | PACKAGE | VERSION | INSTALLED | READY | MESSAGE |
|-------------|---------|---------|-----------|-------|---------|
| redis-cache | redis   | v7.2.0  | True      | True  |         |

To get the full status including conditions, use the following command:

```
d8 k get app -n <NAMESPACE> <APPLICATION_NAME> -o yaml
```

## Conditions

The application state is described in detail through a set of conditions:

| Condition            | Meaning                                                                      |
|----------------------|------------------------------------------------------------------------------|
| Installed            | Package downloaded, manifests and hooks applied for the initial installation |
| UpdateInstalled      | New version downloaded, manifests and hooks applied for the update           |
| ConfigurationApplied | User settings successfully applied                                           |
| Scaled               | All pod replicas are in Ready state                                          |
| Managed              | Application is correctly managed by DKP                                      |
| Ready                | Application is fully operational                                             |

To quickly view all conditions, use the following command:

```
d8 k get app -n <NAMESPACE> <APPLICATION_NAME> \
  -o jsonpath='{range .status.conditions[*]}{.type}: {.status} ({.reason}){"\n"}{end}}'
```

Example output:

```
Installed: True (Installed)
UpdateInstalled: False (Pending)
ConfigurationApplied: True (ConfigurationApplied)
Managed: True (Managed)
Scaled: True (Scaled)
Ready: True (Ready)
```

## Summary

The `status.summary` field provides a brief description of the current application state — useful to check first when diagnosing issues:

```
status:
  summary:
    state: Updating
    message: "Update is waiting for dependent modules to converge; previous version
is still serving"
    tip: "Waiting until DKP processes all dependent modules to start the update."
```

- **state** — current high-level state of the application.
- **message** — explains why the application is in this state.
- **tip** — what to do to resolve the issue or what DKP is waiting for.

## Multiple instances

The same package can be installed multiple times in the same or different namespaces — each with a separate name and settings. For example, two Redis instances can be created — one for caching, one for sessions:

```
# Caching instance
apiVersion: deckhouse.io/v1alpha1
kind: Application
metadata:
  name: redis-cache
  namespace: team-alpha
spec:
  packageName: redis
  packageRepositoryName: my-registry
  packageVersion: "v7.2.0"
  settings:
    maxmemory: "512mb"
---
# Session storage instance
apiVersion: deckhouse.io/v1alpha1
kind: Application
metadata:
  name: redis-sessions
  namespace: team-alpha
spec:
  packageName: redis
  packageRepositoryName: my-registry
  packageVersion: "v7.2.0"
  settings:
    maxmemory: "128mb"
```

All Kubernetes objects created by an Application are prefixed with the instance name — for example, `redis-cache-deployment` and `redis-sessions-deployment` — so names never conflict.

## Updating an application

Updates are manual: change `spec.packageVersion` to the desired version and apply the change:

```
d8 k patch app -n <NAMESPACE> <APPLICATION_NAME> --type=merge -p '{"spec": {"packageVersion": "v7.3.0"}}'
```

While the update is in progress, the `UpdateInstalled` condition will be `False` with `reason: Pending`. Once the update succeeds, it becomes `True`. The previous version continues serving until the update completes.

If the specified version does not exist in the repository, `UpdateInstalled` becomes `False` with `reason: UpdateFailed`, and the current version keeps running.



Specifying an older version (downgrade) is allowed, but DKP does not apply any migration logic on rollback. Verify settings compatibility with the target version before applying the change, if necessary.

## Deleting an application

To delete an application, delete the Application object. For example:

```
d8 k delete app -n <NAMESPACE> <APPLICATION_NAME>
```

When an Application is deleted, all Kubernetes objects it created (unless protected by `helm.sh/resource-policy: keep` or `werf.io/ownership: anyone` annotations in the package templates) will also be deleted.

## FAQ

### Can updates happen automatically?

No. In the current implementation, updates require a manual change to `spec.packageVersion`. Automatic updates via release channels are planned for future versions.

### Can an Application depend on another Application?

No. An Application can only declare dependencies on Modules (via `requirements.modules` in `package.yaml`). This is an architectural constraint that ensures instance isolation.

### Can I install the same application in different namespaces?

Yes. Create Application objects with the same `packageName` and `packageVersion` in different namespaces — each will be a fully independent instance.

## Additional resources

- Global custom resources

## Troubleshooting

### Verify that Marketplace CRDs are present

If `d8 k get app` returns an error, the Marketplace CRDs may not be installed. To check, run the following command:

```
d8 k get crd | grep -E 'application|package'
```

Expected output:



|                                          |                      |
|------------------------------------------|----------------------|
| applicationpackages.deckhouse.io         | 2026-02-10T14:54:41Z |
| applicationpackageversions.deckhouse.io  | 2026-02-10T14:54:41Z |
| applications.deckhouse.io                | 2026-02-10T14:54:41Z |
| packagerepositories.deckhouse.io         | 2026-02-10T14:54:41Z |
| packagerepositoryoperations.deckhouse.io | 2026-02-10T14:54:41Z |

If any CRDs are missing, contact your cluster administrator. Marketplace requires DKP version 1.76 or later.

## Read the application summary

The quickest way to understand why an application is not working is to check `status.summary`. Use the following command (the short name `app` can be used):

```
d8 k get app -n <NAMESPACE> <APPLICATION_NAME> -o yaml | grep -A5 'summary:'
```

Example output:

```
summary:
  state: Updating
  message: "Update is waiting for dependent modules to converge; previous version is
still serving"
  tip: "Waiting until DKP processes all dependent modules to start the update."
```

- **state** — current high-level state of the application.
- **message** — explains why the application is in this state.
- **tip** — what to do to resolve the issue or what DKP is waiting for.

## Read individual conditions

To get a more detailed view of the application state, use the following command:

```
d8 k get app -n <NAMESPACE> <APPLICATION_NAME> \
  -o jsonpath='{range .status.conditions[*]}{.type}: {.status} ({.reason}) -
{.message}}{"\n"}}{end}'
```

Example output showing a stuck update:

```

conditions:
- lastTransitionTime: "2026-02-25T16:39:30Z"
  message: ""
  observedGeneration: 1
  reason: Installed
  status: "True"
  type: Installed
- lastTransitionTime: "2026-02-25T17:12:25Z"
  message: "Update is waiting for dependent modules to converge"
  observedGeneration: 1
  reason: Pending
  status: "False"
  type: UpdateInstalled
- lastTransitionTime: "2026-02-25T16:39:30Z"
  message: ""
  observedGeneration: 1
  reason: ConfigurationApplied
  status: "True"
  type: ConfigurationApplied
- lastTransitionTime: "2026-02-25T16:39:30Z"
  message: ""
  observedGeneration: 1
  reason: Managed
  status: "True"
  type: Managed
- lastTransitionTime: "2026-02-25T16:39:30Z"
  message: ""
  observedGeneration: 1
  reason: Scaled
  status: "True"
  type: Scaled
- lastTransitionTime: "2026-02-25T16:39:30Z"
  message: ""
  observedGeneration: 1
  reason: Ready
  status: "True"
  type: Ready
currentVersion:
version: v0.0.20

```

In this example, `Installed=True` (the application is running on v0.0.20), but `UpdateInstalled=False/Pending` means an update is queued and waiting for a module dependency to settle.

## Check the DKP controller logs

If the status conditions do not provide enough detail for diagnosis, check the controller logs:

```
d8 k logs deployments/deckhouse -n d8-system | grep <APPLICATION_NAME>
```

## Check application pod logs

To list the pods created by the application, run the following command:

```
d8 k get pods -n <NAMESPACE> -l app.kubernetes.io/instance=<APPLICATION_NAME>
```

To view logs for a specific pod, run:

```
d8 k logs -n <NAMESPACE> <POD_NAME>
```

To view logs for a specific deployment prefixed with the instance name, run:

```
d8 k logs -n <NAMESPACE> deployments/<APPLICATION_NAME>--<RESOURCE_NAME>
```

## Common conditions and their meaning

| Condition            | Status=False reason | What to check                                                                                                             |
|----------------------|---------------------|---------------------------------------------------------------------------------------------------------------------------|
| Installed            | InstallFailed       | DKP controller logs, check settings against OpenAPI schema                                                                |
| UpdateInstalled      | Pending             | Dependent module convergence — check d8 module conditions                                                                 |
| UpdateInstalled      | UpdateFailed        | Specified packageVersion does not exist in the repository — verify with <code>d8 k get apv -l package=&lt;name&gt;</code> |
| ConfigurationApplied | ConfigurationFailed | Settings validation error — check against ApplicationPackageVersion schema                                                |
| Scaled               | NotScaled           | Pods not ready — check pod events with <code>d8 k describe pod</code>                                                     |
| Ready                | NotReady            | One or more conditions above are not satisfied                                                                            |

## Additional resources

- Global custom resources

## Configuration of applications logs collection

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 attributes.
- Send logs to multiple storage systems simultaneously (e.g., Loki and Elasticsearch).
- Enrich logs with Kubernetes metadata.
- Use log buffering to improve performance.
- Store logs in internal short-term storage based on Grafana Loki.

The general mechanism of log collection, delivery and filtering is described in detail in the “Architecture” section.

DKP users can configure application log collection parameters using the `PodLoggingConfig` resource, which describes log sources within a specified namespace, including collection, filtering and parsing rules.

## Configuring application log collection

1. Check with the DKP administrator whether log collection and storage are configured in your cluster. Also ask them to provide you with the storage name that you will specify in the `clusterDestinationRefs` parameter.
2. Create a `PodLoggingConfig` resource in your namespace.

In this example, logs are collected from all pods in the specified namespace and sent to short-term storage based on Grafana Loki:

```
apiVersion: deckhouse.io/v1alpha1
kind: PodLoggingConfig
metadata:
  name: app-logs
  namespace: my-namespace
spec:
  clusterDestinationRefs:
    - loki-storage
```

### 3. (Optional) Limit log collection by label.

If you need to collect logs only from specific pods, for example, only from applications with the `app=backend` label, add the `labelSelector` parameter:

```
apiVersion: deckhouse.io/v1alpha1
kind: PodLoggingConfig
metadata:
  name: app-logs
  namespace: my-namespace
spec:
  clusterDestinationRefs:
    - loki-storage
  labelSelector:
    matchLabels:
      app: backend
```

### 4. (Optional) Configure log filtering.

Using `labelFilter` and `logFilter` filters, you can set up filtering by metadata or message fields. For example, in this case, only logs that do not contain fields with the string `.*GET /status" 200$` will be sent to storage:

```
apiVersion: deckhouse.io/v1alpha1
kind: PodLoggingConfig
metadata:
  name: app-logs
  namespace: my-namespace
spec:
  clusterDestinationRefs:
    - loki-storage
  labelSelector:
    matchLabels:
      app: backend
  logFilter:
    - field: message
      operator: NotRegex
      values:
        - .*GET /status" 200$
```

### 5. Apply the created manifest using the following command:

```
d8 k apply -f pod-logging-config.yaml
```

## Examples

### Creating a source in a namespace and reading logs from all pods in it with direction to Loki

The following pipeline creates a source in the `test-whispers` namespace, reads logs from all pods in it and sends them to Loki:

```

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 pods in a specified namespace with a specific label

Example of configuring reading pods with the `app=booking` label in the `test-whispers` namespace:

```

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

```

## Additional resources

- Module `log-shipper` custom resources