Deckhouse Development Portal Documentation

Generated: September 21, 2026

Documentation

Administration guide

Overview

This document is the Administrator's Guide for the Deckhouse Development Portal and is part of the operational documentation for the product.

Installation

Deckhouse Development Portal can be installed in two ways: with external PostgreSQL and Redis instances (connecting to databases already deployed outside the cluster) or with internal instances (deploying PostgreSQL and Redis inside the cluster). External instances are recommended for production; internal instances are suitable for testing and pilot use. Both options are described below.

Installation with internal instances

To install Deckhouse Development Portal, enable the `development-platform` module in your Kubernetes cluster running on Deckhouse Platform. You can use ModuleConfig with minimal settings:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: development-platform
spec:
  enabled: true
  version: 1
  settings:
    rbac:
      superAdminEmail: admin@deckhouse.io
# Super administrator email with full access to portal configuration. Can be changed
at any time.
    security:
      secretKey: "16charssecretkey" # Secret key for encrypting private data. If
changed, API access tokens will need to be regenerated and users will need to re-enter
their credentials.
```

After installation, the Deckhouse Development Portal web UI will be available at `https://ddp.<your domain>`.

When you do not specify `postgres` and `redis` sections, the portal deploys internal PostgreSQL and Redis instances inside the cluster. This scenario is not recommended for production and is suitable only for testing and pilot use; for production, use external resources.

Configuring internal instances (optional)

If you use internal instances, you can explicitly set `mode: internal` and specify images from a private Docker registry:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: development-platform
spec:
  enabled: true
  version: 1
  settings:
    rbac:
      superAdminEmail: admin@deckhouse.io
    security:
      secretKey: "16charssecretkey"
    postgres:
      mode: internal
      image: registry.example.com/postgres:16.3  # PostgreSQL image from private
registry
    redis:
      mode: internal
      image: registry.example.com/redis:7.4.0    # Redis image from private registry.
    additionalImagePullSecrets:
      - "custom-registry-secret"                 # (optional) additional secrets for
private registry access.
```

Installation with external instances

This installation option is recommended for production: the portal connects to PostgreSQL and Redis deployed outside the cluster, which improves resilience and simplifies backup and scaling of databases.

Connecting external PostgreSQL

To use an external PostgreSQL instance, specify connection parameters in the `postgres` section:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: development-platform
spec:
  enabled: true
  version: 1
  settings:
    rbac:
      superAdminEmail: admin@deckhouse.io
    security:
      secretKey: "16charssecretkey"
    postgres:
      mode: external
      host: postgres.example.com  # PostgreSQL server hostname or IP address.
      port: 5432                  # PostgreSQL port (default 5432).
      database: ddp               # Database name.
      username: ddp_user          # Connection username.
      password: secure_password   # Connection password.
```

pg_trgm extension

The portal requires the PostgreSQL `pg_trgm` extension. If you use an external PostgreSQL instance, enable it before starting DDP: connect to the database as a user with permission to create extensions and run:

```
CREATE EXTENSION IF NOT EXISTS pg_trgm;
```

If you deploy the built-in PostgreSQL instance, the extension is created automatically.

Connecting external Redis

To use an external Redis instance, specify connection parameters in the `redis` section:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: development-platform
spec:
  enabled: true
  version: 1
  settings:
    rbac:
      superAdminEmail: admin@deckhouse.io
    security:
      secretKey: "16charssecretkey"
    redis:
      mode: external
      host: redis.example.com       # Redis server hostname or IP address.
      port: 6379                    # Redis port (default 6379).
      database: "0"                 # Redis database index (default "0").
      password: redis_password      # Connection password (optional; leave empty if
Redis has no password).
```

Full example with external instances

Example configuration with external PostgreSQL and Redis:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: development-platform
spec:
  enabled: true
  version: 1
  settings:
    rbac:
      superAdminEmail: admin@deckhouse.io
    security:
      secretKey: "16charssecretkey"
    postgres:
      mode: external
      host: postgres.production.example.com
      port: 5432
      database: ddp
      username: ddp_user
      password: secure_postgres_password
    redis:
      mode: external
      host: redis.production.example.com
      port: 6379
      database: "0"
      password: secure_redis_password
```

Architecture

Security

Role model

The Deckhouse Development Portal (DDP, portal) role model defines which actions users can perform in the portal and which objects they can access. The model controls access to the API and web interface at both the portal and individual object levels.

The role model is implemented in DDP Backend and uses a PostgreSQL database to store roles, permissions, and their relationships.

Role model components

The role model consists of the following components:

- A permission corresponds to a specific action in DDP.
- A role combines a set of permissions.
- A role binding associates a role with users or teams.

Each DDP object has its own permissions, roles, and role bindings. Global roles, permissions, and role bindings allow operations at the platform level.

> Without the required permissions, a user cannot perform operations in the portal. The system checks permissions for every operation.

Object types and permissions

Global permissions

Global permissions apply across the portal and grant access to all objects of a specific type.

Resources:

- `create:resources` — create resources.
- `read:resources` — view resources.
- `update:resources` — edit resources.
- `update:resources-order` — change the order and grouping of resources in the catalog.
- `delete:resources` — delete resources.

Entities:

- `create:entities` — create entities.
- `read:entities` — view entities.
- `update:entities` — edit entities.
- `delete:entities` — delete entities.

Data sources:

- `create:datasources` — create data sources.
- `read:datasources` — view data sources.
- `update:datasources` — edit data sources.
- `sync:datasources` — synchronize data sources.
- `delete:datasources` — delete data sources.

Actions:

- `create:actions` — create actions.
- `read:actions` — view actions.
- `update:actions` — edit actions.
- `run:actions` — run actions.
- `delete:actions` — delete actions.

Automations:

- `create:automations` — create automations.
- `read:automations` — view automations.

- `update:automations` — edit automations.
- `delete:automations` — delete automations.

Workflows:

- `create:workflows` — create workflows.
- `read:workflows` — view workflows.
- `update:workflows` — edit workflows.
- `delete:workflows` — delete workflows.

Webhooks:

- `create:webhooks` — create webhooks.
- `read:webhooks` — view webhooks.
- `update:webhooks` — edit webhooks.
- `delete:webhooks` — delete webhooks.

Widgets:

- `create:widgets` — create widgets.
- `read:widgets` — view widgets.
- `update:widgets` — edit widgets.
- `run:widget-actions` — run widget actions.
- `delete:widgets` — delete widgets.

Dashboards:

- `create:dashboards` — create dashboards.
- `read:dashboards` — view dashboards.
- `update:dashboards` — edit dashboards.
- `delete:dashboards` — delete dashboards.

MCP:

- `read:mcp-servers` — view connected MCP servers.
- `edit:mcp-servers` — connect, edit, synchronize the catalog of, and delete MCP servers.
- `read:mcp-collections` — view MCP collections.
- `edit:mcp-collections` — create, edit, and delete MCP collections.
- `read:mcp-tools` — view custom MCP tools.
- `edit:mcp-tools` — create, edit, and delete custom MCP tools.

External services:

- `create:external-services` — create external services.
- `read:external-services` — view external services.
- `update:external-services` — edit external services.
- `delete:external-services` — delete external services.

Trusted certificates:

- `create:trusted-certificates` — add trusted certificates.
- `read:trusted-certificates` — view trusted certificates.
- `update:trusted-certificates` — edit trusted certificates.
- `delete:trusted-certificates` — delete trusted certificates.

Processes:

- `create:processes` — create processes.
- `read:processes` — view processes.
- `update:processes` — edit processes.
- `delete:processes` — delete processes.

Teams:

- `create:teams` — create teams.
- `update:teams` — edit teams.
- `delete:teams` — delete teams.
- `update:team-variables` — edit team variables.
- `edit:team-filter-rules` — configure group filtering rules during synchronization from Dex.

Icons:

- `create:icons` — create icons.
- `delete:icons` — delete icons.

User credential types:

- `edit:user-access-credentials-types` — create, edit, and delete credential types; configure the Vault integration, including viewing and changing the configuration and testing the connection.
- `rotate:encryption-key` — rotate the encryption key. For details, see Encryption key rotation.

Users:

- `edit:users` — create, block, unblock, and delete users.
- `impersonate:users` — start and end user impersonation. For details, see Impersonation.

> The `update:team-variables` permission allows users to edit variables only for teams of which they are members. Even a super administrator cannot change variables for a team to which they do not belong.

Page view permissions

Permissions with the `view:` prefix control access to sections of the portal interface:

- `view:admin-page` — access the “Administration” section.
- `view:self-service-page` — access the “Self-Service” section.

- `view:ai-page` — access the “AI” section.

> Without the corresponding `view:` permission, a user cannot see a section in the navigation menu even if they have other permissions for objects in that section.

Object-level permissions

Each object type has permissions that apply only to a specific object.

For resources:

- `read:resource` — view a specific resource.
- `update:resource` — edit a specific resource.
- `delete:resource` — delete a specific resource.
- `create:entities` — create resource entities.
- `read:entities` — view resource entities.
- `update:entities` — edit resource entities.
- `delete:entities` — delete resource entities.
- `run:actions` — run actions for resource entities.
- `control:processes` — control processes for resource entities.
- `edit:role-bindings` — edit role bindings for the resource.

For entities:

- `read:entity` — view a specific entity.
- `update:entity` — edit a specific entity.
- `delete:entity` — delete a specific entity.
- `run:actions` — run actions for the entity.
- `control:processes` — control processes for the entity.
- `edit:role-bindings` — edit role bindings for the entity.

MCP collections

For MCP collections:

- `use:mcp-collections` — call collection tools in the AI assistant and through the portal MCP server.
- `edit:role-bindings` — edit role bindings for the collection.

For other objects, including actions, automations, processes, webhooks, widgets, dashboards, and external services:

- `read:[object-type]` — view the object.
- `update:[object-type]` — edit the object.
- `delete:[object-type]` — delete the object.
- `edit:role-bindings` — edit role bindings for the object.

Permission check hierarchy

The role model is hierarchical. When processing a request, DDP Backend checks access rights in a specific order. For example, to determine whether a user can edit a specific entity, the backend performs the following checks:

1. Super administrator

- Check whether the user is a super administrator. If so, stop further checks.

2. Global roles

- Read the user's group membership from the JWT.
- Find global roles for the user and their groups through global role bindings.
- Check permissions in the global roles. If any global role bound to the user or their team contains `update:entities`, the user can edit any entity in DDP, and no further checks are performed.

3. Resource roles

- Find resource roles for the user and their groups through role bindings for the specific resource.
- Check permissions in the resource roles. If any resource role bound to the user or their team contains `update:entities`, the user can edit any entity of that resource, and no further checks are performed.

4. Entity roles

- Find entity roles for the user and their groups through role bindings for the specific entity.
- Check permissions in the entity roles. If any entity role bound to the user or their team contains `update:entity`, the user can edit that entity.

5. Object ownership when ownerIsAdmin is enabled

- Check whether the user owns the object.
- Check whether the user belongs to the team that owns the object.
- If the user owns the object or belongs to its owner team, grant the user all administrator permissions for the object.

If the required permission is not found at any level, the action is denied.

> Ownership is checked only when `ownerIsAdmin` is enabled in the portal configuration. For details, see Object ownership.

Object ownership

Object ownership allows users and teams to own specific system objects and automatically receive all administrator permissions for those objects.

An object owner is a user or team assigned as the owner of a specific object.

Effect of ownership on access rights

When `ownerIsAdmin` is enabled, an object owner receives all administrator permissions for that object. The owner can:

- View, edit, and delete the object.
- Manage role bindings and access for other users.
- Perform any action available to an object administrator.

OwnerAsAdmin option

The `ownerIsAdmin` option controls access rights for object owners.

- “Enabled (`true`)” — object owners receive full administrator permissions for their objects.
- “Disabled (`false`)” — object owners receive no automatic permissions; roles alone control access.

> A portal administrator must configure `ownerIsAdmin` in the configuration file. The option is disabled by default.

Automatic owner assignment on creation

When a user creates an object, such as an action, data source, widget, or resource, the portal automatically assigns the current user as its owner. The owner can be reassigned, or the object can be created without an owner.

Ownership examples

Resource ownership

When a user creates a resource, they automatically become its owner. If `ownerIsAdmin` is enabled, the user receives all administrator permissions for that resource, including:

- Managing resource entities.
- Configuring role bindings.
- Editing and deleting the resource.

Entity ownership

When a user creates an entity, they become its owner and can:

- View and edit the entity.
- Manage role bindings for the entity.
- Delete the entity.

Owner team

If a team owns an object and `ownerIsAdmin` is enabled, all team members receive administrator permissions for the object.

Default role

The portal allows one global role to be set as the default. Its permissions apply to all authenticated users. Configure the default role under “Administration” → “Access control” by using the switch in the “Roles” table.

Only a global role can be set as the default.

Teams

Teams group users and allow roles to be assigned collectively. A user can belong to multiple teams, and their permissions are combined from all teams of which they are a member.

> Teams and team membership are synchronized from the external authentication system, Dex. Teams cannot be managed through the DDP interface.

Configuring roles and managing access

Creating a role

1. Go to “Administration” → “Access control”.
2. Select the “Roles” tab.
3. Click “Create role”.
4. Complete the form:
    - Name — unique role name.
    - Description — role purpose.
    - Object type — select the scope of the role:
        - `Global` — global permissions.
        - `Resources` — resource-level permissions.
        - `Entities` — entity-level permissions.
        - `Actions` — action-level permissions.
        - Other object types.
    - Permissions — select the required permissions.
5. Click “Save”.

Assigning a role to users

1. Go to “Administration” → “Access control”.
2. Select the “Role bindings” tab.
3. Click “Create role binding”.
4. Complete the form:
    - Name — role binding name.
    - Description — role binding purpose.
    - Role — select the role.

    - Object — select a specific object if the role is not global.
    - Users — add the users to whom the role is assigned.
    - Teams — add the teams to which the role is assigned.
5. Click “Save”.

Configuring the default role

1. Go to “Administration” → “Access control”.
2. Select the “Roles” tab.
3. Find the global role to set as the default.
4. Enable the “Default role” switch.
5. Confirm the action.

> Only a global role can be the default.

Editing roles

1. Go to “Administration” → “Access control”.
2. Select the “Roles” tab.
3. Find the required role and click “Edit”.
4. Make the required changes:
    - Change the name or description.
    - Add or remove permissions.
5. Click “Save”.

Editing role bindings

1. Go to “Administration” → “Access control”.
2. Select the “Role bindings” tab.
3. Find the required role binding and click “Edit”.
4. Make the required changes:
    - Change the name or description.
    - Add or remove users.
    - Add or remove teams.
5. Click “Save”.

Deleting roles and bindings

1. Go to the corresponding section.
2. Find the required role or binding.
3. Click “Delete”.
4. Confirm the deletion.

> Deleting a role also deletes all associated role bindings.

Viewing permissions for a user or team

For a user

1. Go to “Administration” → “Users”.
2. Open the target user’s profile.
3. Select the “Permissions” tab.
4. Review:
    - Global permissions.
    - Object-level permissions.
    - Roles assigned to the user.
    - Teams of which the user is a member.

For a team

1. Go to “Administration” → “Teams”.
2. Open the target team’s profile.
3. Select the “Permissions” tab.
4. Review:
    - The team’s global permissions.
    - Object-level permissions.
    - Roles assigned to the team.
    - Users who belong to the team.

> Team membership is synchronized from the external authentication system, Dex.

Role presets

The portal provides role presets for common access scenarios.

Global presets

“Admin” preset

- Type: `Global`.
- Permissions: all global permissions.
- Purpose: system administrators.

“Viewer” preset

- Type: `Global`.
- Permissions:
    - `read:actions`, `read:automations`, `read:dashboards`.
    - `read:datasources`, `read:entities`, `read:external-services`.

    - `read:processes`, `read:resource-relations`, `read:resources`.
    - `read:seeds`, `read:system-alerts`, `read:trusted-certificates`, `read:webhooks`.
    - `read:widgets`, `read:workflows`.
    - `read:audit-logs`, `view:admin-page`, `view:self-service-page`.
- Purpose: users with read-only access.

“Developer” preset

- Type: `Global`.
- Permissions:
    - `read:actions`, `read:dashboards`, `read:external-services`.
    - `read:processes`, `read:widgets`, `read:workflows`.
    - `run:actions`, `run:widget-actions`, `control:processes`.
    - `update:team-variables`.
- Purpose: developers who need to view information and run actions but do not need to create, edit, or delete objects. Developers see only entities to which role bindings at the resource or entity level grant them access.

“Platform engineer” preset

- Type: `Global`.
- Permissions: full access to the catalog and the “Self-Service” page.
- Purpose: engineers who configure the portal, including processes, data sources, and dashboards.

Process presets

“Process admin” preset

- Type: `Processes`.
- Permissions: `delete:process`, `edit:role-bindings`, `read:process`, `update:process`.
- Purpose: process administrators.

“Process viewer” preset

- Type: `Processes`.
- Permissions: `read:process`.
- Purpose: users with permission to view processes.

Resource presets

“Resource admin” preset

- Type: `Resources`.
- Permissions: all resource permissions.
- Purpose: administrators of specific resources.

Using presets

1. Go to “Administration” → “Access control”.
2. Select the “Roles” tab.
3. Click “Create role”.
4. Select an object type.
5. Under “Presets”, click the required preset.
6. Configure the role:
    - Change the name and description if necessary.
    - Add or remove permissions.
7. Click “Save”.

Role configuration examples

“System administrator” role

- Type: `Global`.
- Permissions: all global permissions. Use the “Admin” preset.
- Assignment: super administrators.

“Resource administrator” role

- Type: `Resources`.
- Permissions: all resource permissions. Use the “Resource admin” preset.
- Assignment: a specific resource and the “Administrators” team.

“Process operator” role

- Type: `Processes`.
- Permissions: `delete:process`, `edit:role-bindings`, `read:process`, `update:process`. Use the “Process admin” preset.
- Assignment: the “Process operators” team.

Best practices

Creating roles

1. Determine the access level: decide whether you need a global role or a role for a specific object.
2. Select the required permissions according to the principle of least privilege.
3. Create the role in the corresponding administration section.
4. Assign the role to users or teams through role bindings.

Managing access

1. Use teams to manage access for groups.
2. Use default roles to grant basic permissions to all users.
3. Follow the hierarchy: use global roles for broad access and object roles for specific access.
4. Regularly review assigned roles and bindings.

Security

1. Apply the principle of least privilege: grant only the required permissions.
2. Audit regularly: review assigned roles and their use.
3. Use teams instead of individual assignments to simplify management.
4. Document roles with descriptions that explain their purpose.

Impersonation

DDP (portal) supports impersonation: a user with the global `impersonate:users` permission can temporarily use the web interface as another user.

> Impersonation is available only for browser sign-in through a Dex session. API tokens are not supported.

Starting and ending impersonation

1. Go to “Administration” → “Users”.
2. In the target user’s row, click “Sign in as this user”.
3. The portal switches the session to the selected user. A “Signed in as: …” banner at the bottom of the screen shows the time remaining before the session ends automatically.
4. To end impersonation early, click “End session” in the banner.

The “Sign in as this user” button is unavailable for blocked users and your own account.

Security restrictions

Impersonation does not start in the following cases:

- The selected user is blocked.
- The selected user is the user who is already signed in.
- The selected user is a super administrator, but the operator is not a super administrator.
- The selected user already has the global `impersonate:users` permission, unless the operator is a super administrator.

Session lifetime

An impersonation session has a limited lifetime:

- The session lasts 30 minutes.
- The session ends automatically when it expires.
- The web interface displays a countdown until automatic termination.

If the user does not end impersonation manually, the portal ends it automatically when the session expires.

Audit

For HTTP requests included in audit logs (`POST`, `PUT`, `DELETE`, and `PATCH`), the “Email” field identifies the operator and target user in the format `operator@example.com [as target@example.com]`.

This format lets you determine the following from an audit log entry:

- Who initiated the request.
- Whose identity the session used to perform the operation.
- The HTTP method, path, and response status.

`GET` requests are not recorded in audit logs. For details, see Audit logs.

Encryption key rotation

Deckhouse Development Portal (DDP) lets you securely replace the encryption key (`security.secretKey`) without losing credentials or other encrypted values. The operation runs from the web interface and re-encrypts all data stored with the current key.

Reasons to rotate the key

Rotate the encryption key when you need to replace `security.secretKey` in the DDP Backend configuration, for example, to comply with a security policy or after the key has been compromised.

> ⚠ Do not change `security.secretKey` in the configuration or restart the backend before re-encryption finishes in the web interface. Otherwise, the stored values will become unavailable.

Access permissions

Key rotation requires the global `rotate:encryption-key` permission.

Data that is re-encrypted

The operation affects the following data categories:

- “User credentials” — values in PostgreSQL for credential types that use the “Database” storage.
- “AI provider credentials” — personal credentials for AI integrations.
- “Temporary action responses” — encrypted temporary responses in action execution records.
- “Vault secrets” — credential values stored in HashiCorp Vault or Deckhouse Stronghold.
- “Vault configuration” — the AppRole Role ID and AppRole Secret ID in the Vault integration settings.

Procedure

1. Go to “Administration” → “Credentials”.
2. Click “Rotate encryption key”.

3. In the “Old encryption key” field, enter the current key from the DDP configuration (`security.secretKey`).
4. In the “New encryption key” field, enter the new key. It must meet the same requirements as during initial setup:
    - The key must be 16, 24, or 32 bytes long.
    - The key may contain only printable ASCII characters.
    - The key must not consist of a single repeated character.
5. Click “Re-encrypt”.
6. Wait for the operation to finish and check the results table. For each category, the table shows the number of successfully re-encrypted, skipped, and failed records.
7. Set `security.secretKey` in the DDP Backend configuration to the new key.
8. Restart DDP Backend.

> After re-encryption, credentials remain unavailable until you specify the new key in the backend configuration and restart the service.

Operation results

The results table shows three counters for each data category:

- “Successful” — records re-encrypted with the new key.
- “Skipped” — records already accessible with the new key, for example, records re-encrypted earlier.
- “Failed” — records that could not be re-encrypted, usually because the old key is incorrect or the data is corrupted.

The “Total” row shows totals across all categories.

If the “Failed” column contains non-zero values, do not update the backend configuration until you resolve the errors. Verify the old key and repeat the operation.

Audit logs

Audit logs record all operations that users perform through the Deckhouse Development Portal (DDP) API. The logs are stored in a PostgreSQL database and support auditing of user activity.

Components

The following components create and store audit logs:

- DDP Backend creates audit log entries.
- PostgreSQL stores the data.

Audit log contents

Each HTTP request except `GET` creates an audit log entry with the following fields:

- “Email” — email address of the user who made the request.
- “IP” — client IP address.
- “Request body” — HTTP request body.
- “Path” — requested API path.
- “Method” — request HTTP method (`POST` , `PUT` , `DELETE` , or `PATCH`).
- “Status” — response HTTP status code.
- “Date” — request date.

How it works

DDP Backend automatically creates audit logs through request-processing middleware for all HTTP requests except `GET` .

To reduce database load, entries are buffered with the following settings:

- Batch size — 40 entries.
- Write interval — 20 seconds.
- In-memory buffer size — 4 entries.

Storage

Audit logs are stored in the PostgreSQL `audit_logs` table. The table is partitioned by day based on the `timestamp` field.

The table structure is maintained automatically. Every day at `00:00` server time, the system prepares space for logs for the next 7 days.

Logs are retained indefinitely and old entries are not deleted automatically. To control retention, configure external tools, such as cron jobs or database cleanup scripts, to delete old log partitions automatically.

Viewing logs

Audit logs are available in the web interface under “Administration” → “Audit”. You can filter logs by the following fields:

- Period, including start and end date and time.
- User email.
- IP address.
- Path.
- Request method.
- Response status.

Exporting to CSV

The audit log page provides a “Download .csv” button. Export is available only with permission to read audit logs.

The file includes all entries that match the current filters and sorting. Export does not run if more than 100,000 events match.

Configuration

Audit logging is enabled by default and requires no additional configuration. To control log retention, configure automatic deletion of old database partitions.

Actions

Overview

Actions are a DDP (portal) mechanism for running operations in external infrastructure systems and services. For example, actions can:

- create projects, variables, branches, tags, releases, and merge requests in GitLab;
- create resources in Kubernetes and retrieve them;
- create secrets in Deckhouse Stronghold and HashiCorp Vault, as well as clients in Keycloak;
- create projects in SonarQube, products and engagements in DefectDojo, projects in CodeScoring, and repositories, roles, users, and privileges in Nexus Repository;
- create topics and ACLs in Apache Kafka;
- perform helper actions for debugging and controlling process execution (for example, Debug and Wait).

An action can be linked to one or more resources. Once linked, it can be run for any entity of those resources.

Action configuration

Basic information

When creating or editing an action, specify the basic information:

Field	Required	Description
Name	Yes	Arbitrary name of the action
Identifier	Yes	Action identifier. Generated automatically from the name
Resource	No	One or more resources for which the action will be available to run
Icon	No	Icon displayed on the action card
Owner	No	User account responsible for the action’s configuration and operation
Owner team	No	Team responsible for the action’s configuration and operation
Tags	No	Tags used to classify and search for actions
Description	No	Description in Markdown. Displayed when running the action or a process that contains it

Request configuration

Action type

An action can be of the following types:

- “Built-in (BuiltIn)” — the action’s execution logic is defined within the portal. For built-in actions, you must select one of the preconfigured backends.
- “Webhook” — the action’s execution logic is fully configured by the user.

Retry parameters

If an action fails, the portal can automatically retry it.

Number of retries

How many times to retry execution after a failure. 0 means a single attempt with no retries.

Base delay (sec.)

The delay in seconds before the first retry; the pause doubles before each subsequent attempt (exponential backoff).

Request body format

For webhook actions, you can choose the format used to send the request body:

- JSON (default) — the request body is sent in JSON format with the `Content-Type: application/json` header.
- "Form URL Encoded" — the request body is sent in `application/x-www-form-urlencoded` format.

> When "Form URL Encoded" is selected, the request body must contain only flat key-value pairs. Nested structures and arrays are not supported in this format.

Example:

```
token: {{ .credentials.token }}
```

Extended logging

For webhook actions, you can enable an extended logging option that provides detailed logging of all HTTP request and response details:

- request URL;
- HTTP method;
- HTTP headers (including tokens);
- request body;
- response status code;
- response HTTP headers;
- response body.

When this option is enabled, all this data is written to the action run log. When it is disabled, logging works in standard mode.

> Enabling extended logging may result in sensitive information (tokens, passwords, etc.) being written to the logs. Use this option with caution and only when needed for debugging.

Request body

Each action sends an HTTP request to a built-in backend or to a webhook backend URL. The request usually includes a request body (`body`) that describes what exactly will be sent. The request body is defined in YAML.

Parameter values from the user form can be substituted into the request body using Go template templates.

Example:

```
project_id: {{ .property.project_id }}
```

With this request body, the expression `{{ .property.project_id }}` is replaced with the value of the `project_id` parameter that the user entered in the action's launch form.

For built-in actions, the "Request body" parameter is part of the action configuration, and a configuration example is available.

User form

Parameters

The "User form" section specifies the parameters that the user can fill in when launching the action. Parameter types are described in the Parameters section.

The following options are available for each parameter:

- "Editable" — allows the parameter value to be changed when launching the action. If this option is disabled, the value cannot be changed.
- "Required" — requires a value to be specified when launching the action.
- "Hidden" — the parameter is not displayed in the user form when launching the action.

Each parameter can have a default value that is pre-filled in the form when the action is launched.

> For non-editable or hidden parameters, it is recommended to set a default value, since the user will not be able to change them when launching the action.

The parameter description is shown when launching the action by clicking the "info" icon. Providing a description makes the parameter's purpose easier to understand and reduces the risk of errors when launching the action.

Parameter values can use Go template template functions. For example, the expression `{{ .entity.name }}` in a parameter value means that when the action is launched, the name of the entity it is running for will be substituted.

Parameter conditions

Parameters can be automatically hidden or shown in the user form depending on the value of a `Boolean`-type parameter. This is configured in the "Parameter conditions" section, where you define rules for showing or hiding selected parameters.

Update

Templating contexts

All updates are performed only after the action completes successfully. The following templating contexts are available in Go templates:

Context	Description	Example
`.property`	Values from the user form	`{{ .property.repo }}`
`.response`	Parsed action response (JSON)	`{{ .response.id }}`
`.entity`	Parameters of the catalog entity for which the action is run	`{{ .entity.slug }}`
`.workflow`	Workflow parameters	`{{ .workflow.branch }}`
`.process`	Process parameters	`{{ .process.releaseId }}`
`.global`	Global variables (variable set identifier, then key)	`{{ .global.myGroup.apiUrl }}`
`.team`	Variables of the team selected at launch	`{{ .team.token }}`
`.store`	Variables from the process store	`{{ .store.myKey }}`

Entity parameter update

If the “Update entity parameters” option is enabled, the portal applies the update rules and writes the values to the entity’s parameters.

Field	Description
Source	Go template for the value
Entity parameter	Identifier of the entity parameter in the catalog

For example, when the “Create GitLab project” action is run, the response (`response`) will contain the specification of the created project:

```
{
  "id": "1",
  "...": "..."
}
```

If you need to immediately populate the entity's `repository_id` parameter after creating a project in GitLab, specify the following update rule:

1. Source: `{{ .response.id }}`.
2. Entity parameter: `repository_id`.

Entity creation

If the "Create entities" option is enabled, the portal automatically creates new entities in the selected resources according to the specified rules. Rules are defined separately for each resource.

Field	Description
Owner of created entities	Assigned as the owner of all entities the action creates in the catalog
Owner team of created entities	Assigned as the owner team of all entities the action creates in the catalog
Resource	Catalog resource in which the entity will be created
Source (identifier)	Go template for the identifier of the entity being created
Source (name)	Go template for the name of the entity being created
Additional rules	Mapping of a Go template value to an entity parameter

The identifier and name are required for each rule group. If a resource already has an entity with the same identifier, creation is skipped.

For example, when the "Create GitLab project" action is run, the response may contain:

```
{
  "id": "42",
  "name": "my-project",
  "...": "..."
}
```

To create an entity in the “GitLab projects” resource after the action completes successfully, specify the following rules:

1. Resource: “GitLab projects”.
2. Source (identifier): `{{ .response.id }}`.
3. Source (name): `{{ .response.name }}`.

If needed, add additional rules to populate entity parameters, for example map `{{ .response.path }}` to the `path` parameter.

Entity relation creation

If the “Create entity relations” option is enabled, the portal automatically creates new relations for the selected entity according to the specified rules. The set of rules is defined separately for each resource.

Field	Description
Resource	Either of the two resources involved in the required resource relation: the list of relations in which the selected resource is specified as the parent or the child is loaded based on the selected resource. You can choose the resource of the entity for which the action is running, or the resource on the other side of the relation — the list will contain the same relation if both resources are part of it
Relation	The catalog resource relation: it defines the parent and child resources and entity roles when creating the relation. The selected relation determines which side the launch entity is on and which of the fields below specifies the identifier of the second entity from the action’s response
Parent entity identifier	Go template for the identifier of the parent entity in the catalog
Child entity identifier	Go template for the identifier of the child entity in the catalog

User credentials update

If the user credentials update option is enabled, the action automatically updates the user’s credentials according to the specified rules.

Field	Description
Source	Go template for the value
Credentials type	Type of credentials to be updated

For example, when running an action that returns a new API key in its response:

```
{
  "apiKey": "new-api-key-12345",
  "...": "..."
}
```

To update the credentials, specify the following rules:

1. Source: `{{ .response.apiKey }}`.
2. Credentials type: select the type of credentials to be updated.

Selecting a user for credentials update

By default, credentials are updated for the user who launched the action (the initiator).

To update the credentials of a different user, enable the “Update credentials of a specific user” option and select the desired user.

Process store update

Rules from the “Process store update” section only apply when the action is run as part of a process: after the action completes successfully, values are written to the run’s store according to the specified rules, in list order.

Field	Description
Condition	An optional Go template; if empty, the rule always applies. Rendering result: `true`, `false`, `1`, or `0`
Operation	Write string, Write JSON, Append string, Append JSON, Merge JSON (shallow), Delete
Target	Dot-path in the store without a leading dot, e.g. `deploy.job_id` or `notification.items`
Source	Go template for the value

Other actions and process elements read this data via `{{ .store.<path> }}`. Inside a loop, the `{{ .store._loop.* }}` context is available in templates.

If there are no rules (the list is empty), nothing is written to the store when the action runs as part of a process.

For a detailed description of operations, rule examples, iterating over a collection, and viewing the store, see Process store.

Security

Execution conditions

The action will only run if the entity's status matches one of the selected values in the status field. If no restrictions are set, this condition does not apply.

Masking action fields

For each action, you can enable masking of fields that may contain sensitive information.

If this option is enabled, the following will be hidden in the action's run records:

- values of all filled-in parameters;
- request body (`body`);
- response (`response`) generated as a result of execution.

Temporary response

For each action, you can enable the "Temporary response" option. It restricts access to the action's execution result and hides it from other users.

If this option is enabled, after the action completes successfully, the response:

- is stored as temporary and displayed only to the user who launched the action;
- is removed from the regular response field so that it is not accessible to other users.

Other users will see only an encrypted placeholder instead of the response content.

The action's initiator can delete the temporary response manually.

> The temporary response is available only to the action's initiator. Other users cannot view its contents and cannot delete it.

Approval

You can enable mandatory approval for an action and specify the number of approvers required. In this case, the action will not run until the specified number of approvals has been received from the designated users.

The current number of approvals and the list of users whose approval is expected are displayed in the action run table for the corresponding entity.

If approval is required from a specific user, they will receive a notification in the web interface.

Approval notifications

When approval is enabled, you can enable notifications and specify how they are sent. If sending to a URL is selected, specify the webhook address, optionally disable the server’s TLS certificate verification, and add additional HTTP request headers.

Authorization

Use external service

Select the external services for which the action can be run. If multiple external services are added, a specific service can be selected when launching the action.

URL and HTTP headers

Specify the address to which the request will be sent and the HTTP headers.

Headers support Go templates in the form `{{ .credentials.<credentials type identifier> }}` for substituting user credentials.

Select an account to run as

By default, external infrastructure services are accessed using the credentials of the user who launched the action. If needed, you can explicitly specify that the action should run on behalf of a specific account.

For actions launched as automation events, specifying the account to run as is mandatory.

Credentials

For built-in actions, the portal predefines the set of required credentials. Their identifiers are loaded when the built-in backend is selected. For each identifier, you must select the type of credentials to be used.

For webhook actions, credentials can be accessed in the request body using the `{{ .credentials.<credentials type identifier> }}` construct.

HTTP method

For webhook actions, the HTTP method of the outgoing request is specified.

Action runs

Action run records

Each time an action is run, a record is created containing the initiator, execution status, and a detailed log. Run records for each entity are available on the entity’s card, on the “Action runs” tab. If the action requires approval, it is handled on the same tab.

A run record can be deleted, or the action can be re-run. Re-running creates a new record.

Run logging

For each action run, a record containing the full execution log is created in the database.

The `actions.logging.enabled` parameter in the DDP configuration file controls whether run logs are output to `stdout`: when set to `true`, logs are output; when set to `false`, they are not.

> Records with the full run log are created in the database regardless of the value of `actions.logging.enabled`.

Action run statuses

For each action run, a record with a status is created. Possible statuses:

- `Created` — the record has been created, but the action has not been run yet.
- `Unapproved` — the action is awaiting approval.
- `Running` — the action is running.
- `Failed` — the action finished with an error.
- `Update failed` — the action completed, but updating the entity's parameters failed.
- `Success` — the action completed successfully.
- `Retrying` — the action finished with an error and is being retried.

GitLab

CreateRepositoryFromTemplate

> Running this action requires credentials:
> - `password` — the password (token) for the user on whose behalf the action will run.
> - `username` — the username of the user on whose behalf the action will run.

CreateRepositoryFromTemplate — creates a new repository from a template in GitLab. The rendering mechanism is based on Go template and supports all built-in methods, as well as extensions added by DDP (portal).

Request example

```
sourceBranch: main
sourceTag: v1.0.0
templateRepositoryUrl: https://gitlab.example.com/example-1.git
targetRepositoryUrl: https://gitlab.example.com/example-2.git
targetBranch: master
additionalIgnoreFiles:
  - .ignore
  - .example
values:
  key1: value1
  nested:
    enabled: true
    subkey: 123
```

Request specification

Name	Required	Description	Default value
templateRepositoryUrl	Yes	URL of the template repository	-
targetRepositoryUrl	Yes	URL of the repository to be created as a result of running the action	-
values	Yes	Variables used during templating, in `key: value` format	-
additionalIgnoreFiles	No	List of files containing paths to exclude from the target repository. Populated similarly to .templateignore	-
sourceTag	No	Tag of the template repository to use during templating. If not specified, the template repository's branch is used	-
sourceBranch	No	Branch of the template repository to use during templating	main
targetBranch	No	Branch of the target repository to be created as a result of running the action	main

How it works

The portal:

1. Clones the template repository from the specified URL (`templateRepositoryUrl`), using `sourceTag` , `sourceBranch` , or the `main` branch as the ref, in that order of preference.
2. Reads the `values.yaml` file stored at the root of the repository and determines the default templating variables.

3. Reads the variables passed when the action is launched and merges them with the variables from `values.yaml`. Variables passed at launch take priority during the merge.
4. Reads the `.templateignore` file and determines the directories and files excluded from templating.
5. Renders the files from the templates, taking into account `values.yaml` and the variables passed to the action.
6. Changes the repository remote to the target repository (`targetRepositoryUrl`) and pushes to the target branch (`targetBranch`) or to the `main` branch.

Implementation details

The action supports templating of directory and file names. To do this, add an expression in Go template format to their name. For example, the directory `src/{{ .module }}/utils`, given a `module` value of `example`, will be rendered as the directory `src/example/utils` in the target repository.

If, after rendering from the template, a file's content is empty, the file is not created. For example, a file with the following content:

```
{{- if .createContent }}
- This is content that will be displayed if the createContent variable == true
{{- end }}
```

will not be created if the `createContent` variable is `false`. Similarly, files that are originally empty will not be created either.

If templating variables are missing from both the `values.yaml` file and the variables passed when the action is launched, rendering fails with an error and the target repository is not created.

Template repository variables

To add default variables used during templating, create a `values.yaml` file with the appropriate content at the root of the repository.

Example `values.yaml` file:

```
module: example
createContent: false
```

The `values.yaml` file is optional.

Excluding files

Some files may contain variables in Go template format that need to be preserved when rendering the repository from a template, for example Helm charts in the `helm` directory. The `.git` directory is always ignored.

To exclude such files from the rendering mechanism, add a `.templateignore` file with the appropriate content to the root of the repository.

Each line of `.templateignore` defines a single rule — a path or a mask. If the line contains `{{`, the entire line is executed as a single Go template with the same variables and functions used when substituting into file and directory names (for more information, see How a rule line is processed). If the line does not contain `{{`, no substitution of variables from `values.yaml` or from the action's request is performed: the line is read as-is and compared with the relative path on disk; literals and the mask characters `*` and `**` are allowed.

Example of a `.templateignore` file containing only path masks, without substitution templates, to ignore the contents of the `helm` and `docs` directories:

```
helm/**
docs/**
```

Adding paths to ignore

1. At the root of the template repository, create or edit the `.templateignore` file (one rule per line).
2. Each line is a single rule: a path from the repository root. Regular path characters and masks are allowed: an asterisk `*` within a segment name, and the sequence `**` for arbitrary directory nesting. The portal matches the relative path against the mask according to built-in rules (similar to common conventions used for masks in ignore files in version control systems).
3. To substitute a path fragment from `values.yaml` or from the `values` field of the action's request, use Go template constructs (`{{ ... }}`) in the line. If the line contains `{{`, the portal processes the entire line from start to finish as a single Go template: you cannot leave part of the line as "plain text" and template only the middle of the path. See examples in Go template examples in .templateignore.
4. Empty lines and lines starting with `#` are skipped when parsing the file — you can use them for comments.

Examples without substitution (path masks only)

Individual files at the root:

```
package-lock.json
yarn.lock
LICENSE
.env.local
```

Entire directories and typical build artifacts:

```
vendor/**
node_modules/**
dist/**
build/tmp/**
```

Nesting with a mask:

```
docs/**/*.pdf
charts/*/values.schema.json
.github/workflows/**
```

Secrets by extension across the entire tree:

```
**/*.pem
**/*.key
```

How a rule line is processed

The variables available for expanding rules are the same ones merged from `values.yaml` at the root of the template repository and from the `values` field in the action's request (the `values` field in the request takes priority).

For each non-empty line from `.templateignore` or from a file listed in `additionalIgnoreFiles`, the portal does the following.

1. The line is always added to the rule list exactly as it appears in the file, unchanged. This is the line that is compared with the file path first — this is needed for cases where the on-disk names still contain fragments like `{{ .module }}` before renaming.
2. If the line contains `{{`, the portal runs the entire line once through the Go template engine, with the same capabilities used when substituting into file and directory names (built-in portal functions and the Sprig set). If the line does not contain `{{`, this step is skipped.
3. If the substitution step was performed and the resulting text differs from the line in the file (including the case where the result is empty), a second entry is added to the rule list — with this resulting text. As a result, a single line in the file can produce two entries in the list. When checking a file path, both are checked: a match with either one is enough for the file to be covered by the rule.

Then, while traversing the tree, for each file path, the path relative to the root of the cloned copy is calculated (with forward slashes). The path is compared against each rule: first using mask-matching rules (including `*` and `**`), and, if necessary, by an exact match between the rule string and the relative path.

This way, a single line in the file can define two rules: for example, the original `charts/{{ .name }}/**` (so as not to affect the path before directories are renamed), and the expanded `charts/billing/**` (to

match the path after variable substitution into on-disk names). This is why the rules work both “before” and “after” the directory-renaming step of the template.

If the template in a line has a syntax error or references a missing field, rule expansion fails with an error, and repository rendering does not continue.

The `additionalIgnoreFiles` field in the action specifies the names of additional files at the root of the repository; each of them contains rule lines just like `.templateignore`, with the same template expansion. However, the meaning is different: matched paths are removed from the working copy (the directory or file is deleted), and this is done twice — at the beginning and at the end of the processing chain — so that these objects do not participate in further steps and do not end up in the resulting repository.

The `.templateignore` file, on the other hand, means “do not template”: for matched paths, the portal does not substitute variables into file contents and does not apply template-based renaming to the corresponding directory paths. The files and directories themselves are not deleted just because of an entry in `.templateignore` — they remain in the copy, but are processed as plain text and plain names, without the templating step.

Go template examples in .templateignore

Each example below shows fragments of `values.yaml` (or the equivalent fields in the request’s `values`) and lines from `.templateignore`. Several independent rules are defined by several lines in the file: the result of one template line is one masked line; line breaks within the template’s result do not split the rule into several.

The directory name in the rule comes from `values.yaml` or from the action’s `values`:

`values.yaml`:

```
project: payment-gateway
```

`.templateignore`:

```
{{ .project }}/legacy/**
```

The rule set will include the lines `{{ .project }}/legacy/**` and `payment-gateway/legacy/**`.

A path segment from a variable and a fixed part that is preserved after the variable’s value is substituted:

`values.yaml`:

```
lang: ru
```

`.templateignore`:

```
apps/{{ .lang }}/messages.yaml
```

A conditional rule on a single line works as follows: when `skipGenerated: false`, the substitution produces an empty string, which is still added as a second rule. The original line with `{{` is used as a path mask and usually does not match any real paths. When `skipGenerated: true`, the second rule becomes `generated/**`:

`values.yaml`:

```
skipGenerated: false
```

`.templateignore`:

```
{{- if .skipGenerated }}generated/**{{- end }}
```

A variant for “ignore only non-production”:

`values.yaml`:

```
tier: staging
```

`.templateignore`:

```
{{- if ne .tier "prod" }}mock/**{{- end }}
```

Using `printf` to build a mask string (convenient when the chart name is a variable):

`values.yaml`:

```
chartName: wordpress
```

`.templateignore`:

```
{{ printf "charts/%s/**" .chartName }}
```

A default value for an “empty” value — the `default` function from the Sprig set. The field must exist in the data (otherwise expansion will fail with `missingkey=error`); for “not set in YAML”, add a key with an empty string, or use the `if` / `index` construct:

`values.yaml`:

```
envName: ""
```

`.templateignore`:

```
{{ default "dev" .envName }}/secrets/**
```

With an empty `envName`, the rule set will include both `{{ default "dev" .envName }}/secrets/**` and `dev/secrets/**`.

An optional path segment (`with`):

`values.yaml`:

```
analyticsModule: tracking
```

`.templateignore`:

```
{{ with .analyticsModule }}{{ . }}/vendor/**{{ end }}
```

If `analyticsModule` is empty, the template produces an empty string (the expansion rules are described above).

Accessing a nested structure's field by a string key (`index`):

`values.yaml`:

```
regions:
  primary: eu-west
```

`.templateignore`:

```
configs/{{ index .regions "primary" }}/bootstrap.yaml
```

Trimming whitespace from a name segment — the `trim` function from the Sprig set:

`values.yaml`:

```
serviceName: " billing-api "
```

`.templateignore`:

```
{{ trim .serviceName " " }}/logs/**
```

Several fragments in a single mask:

`values.yaml`:

```
base: services
variant: canary
```

`.templateignore`:

```
{{ .base }}/{{ .variant }}/**/*.tmp
```

Examples for additionalIgnoreFiles

The action's specification lists the names of files at the root of the repository (for example, `.ship-ignore`, `.ci-remove`). The line format inside such files is the same: `#` comments, empty lines, path masks, and, if needed, Go templates in lines.

The `.ship-ignore` file in the template:

```
# excluded from the target repository
local/fixtures/**
scratchpad.md
```

Fragment of the action's request:

```
additionalIgnoreFiles:
  - .ship-ignore
```

A template in a file used with `additionalIgnoreFiles` (removing a directory, with the name coming from variables):

The `.env-drop` file at the root of the template:

```
{{ .obsoleteDir }}/**
```

With `obsoleteDir: legacy-ui` from `values`, after expansion, the deletion list will contain both `{{ .obsoleteDir }}/**` and `legacy-ui/**` — matching paths will be removed from the copy before the final steps.

> Note the difference:
> - `.templateignore` leaves files on disk but disables path renaming and content rendering for them;
> - lists from `additionalIgnoreFiles` remove matched paths from the working copy.

Example directory structure of a template repository

```
├── example-folder-01
│   ├── example-file-01
│   └── {{ .example }}-file-02
├── {{ .example }}-folder-02
│   └── ...
├── values.yaml
└── .templateignore
```

If the `example` variable is set to `new` when the repository is rendered, the resulting structure after rendering will look as follows:

```
├── example-folder-01
│   ├── example-file-01
│   └── new-file-02
├── new-folder-02
│   └── ...
├── values.yaml
└── .templateignore
```

Local debugging

The `ddp-render-dir` utility is available for local debugging of templates.

The utility:

1. Creates a copy of the source directory.
2. Renders the files in this directory using the same rules as the action that creates repositories from templates.

Command-line flags for running it:

- `--source-dir` — the source directory to render.
- `--target-dir` — the directory where the rendering result will be placed.
- `--values` (optional) — path to a `values.yaml` file with variables to use during rendering.
- `--ignore-files` (optional) — a list of files containing paths to exclude from the target repository.

CreateGitlabProject

> This action requires a token for the user on whose behalf it will run.

CreateGitlabProject — creates a new project in GitLab through the GitLab API, using the specified name, project path, description, and other settings. Authentication uses a GitLab token provided in the credentials.

Request example

```
name: example
path: example
description: example
default_branch: main
initialize_with_readme: false
namespace_id: '0'
```

Request specification

Name	Required	Description	Default value
name	Yes	Name of the project to be created in GitLab	-
path	Yes	URL-friendly project path. Usually matches the name, but may differ	-
default_branch	Yes	Name of the branch to be used by default, e.g. “main”	-
namespace_id	Yes	Identifier of the GitLab namespace in which the project will be created	-
initialize_with_readme	No	Flag indicating whether the project should be initialized with a README file	false
description	No	Project description visible to users	-

Note

The action performs a POST request to the URL: `/api/v4/projects`, sending the request parameters in JSON format. Upon successful creation, GitLab returns data about the newly created project.

CreateGitlabProjectWebhook

> This action requires a token for the user on whose behalf it will run.

CreateGitlabProjectWebhook — creates a webhook in a GitLab project.

Request example

```
project_id: '0'
url: https://example.com
push_events: true
issues_events: true
merge_requests_events: true
pipeline_events: true
```

Request specification

Name	Required	Description
project_id	Yes	Identifier of the project in which to create the webhook
url	Yes	Webhook URL
push_events	Yes	Trigger the webhook when changes are pushed to the repository
issues_events	Yes	Trigger the webhook when an issue is created
merge_requests_events	Yes	Trigger the webhook when a merge request is created
pipeline_events	Yes	Trigger the webhook when a pipeline is run

Note

The action performs a POST request to the URL: `/api/v4/projects/:id/hooks`.

CreateGitlabBranches

> This action requires a token for the user on whose behalf it will run.

CreateGitlabBranches — creates new branches in the target repository.

Request example

```
project_id: '0'
branches:
  - branch: new-branch
    ref: main
```

Request specification

Name	Required	Description
project_id	Yes	Identifier of the project in which to create the branches
branches	Yes	List of branches to create
branches.branch	Yes	Name of the new branch
branches.ref	Yes	Name of an existing branch or a commit SHA

Note

The action performs a POST request to the URL: `/api/v4/projects/:id/repository/branches`. Upon successful creation, GitLab returns information about the created branches.

CreateGitlabTag

> This action requires a token for the user on whose behalf it will run.

CreateGitlabTag — creates a new tag in a GitLab project.

Request example

```
project_id: '0'
tag_name: v1.0.0
ref: main
message: Tag description
```

Request specification

Name	Required	Description
project_id	Yes	Identifier of the project in which to create the tag
tag_name	Yes	Name of the tag
ref	Yes	Name of the branch or tag, or the commit SHA, that the new tag will point to
message	Yes	Tag description

Note

The action performs a POST request to the URL: `/api/v4/projects/:id/repository/tags`.

CreateGitlabRelease

> This action requires a token for the user on whose behalf it will run.

CreateGitlabRelease — creates a GitLab release based on an existing tag.

Request example

```
project_id: '0'
tag_name: v1.0.0
name: Release v1.0.0
description: |
  ## Changes:
  - New feature.
  - Bug fixes.
```

Request specification

Name	Required	Description
project_id	Yes	Identifier of the project in which to create the release
tag_name	Yes	Name of the existing tag on which the release will be based
name	Yes	Name of the release as displayed in GitLab
description	No	Release description in Markdown format

Note

The action performs a POST request to the URL: `/api/v4/projects/:id/releases`.

CreateGitlabMergeRequest

> Running this action requires credentials:
> - `password` — the password (token) for the user on whose behalf the action will run.
> - `username` — the username of the user on whose behalf the action will run.

CreateGitlabMergeRequest — creates a new Merge Request (MR) in the target repository. Files stored in the source repository are added to the Merge Request. Files may contain variables whose values will be substituted at the time the MR is created.

Request example

```
source_project_id: '0'
source_project_branch: example
source_project_tag: v1.0.0
target_project_id: '0'
merge_request_spec:
  source_branch: example
  target_branch: '1'
  title: example
additionalIgnoreFiles:
  - .ignore
  - .example
values:
  key1: value1
  nested:
    enabled: true
    subkey: 123
```

Request specification

Name	Required	Description	Default value
source_project_id	Yes	Identifier of the project that serves as the source for the Merge Request	-
target_project_id	Yes	Identifier of the target project in which the Merge Request will be created	-
merge_request_spec	Yes	Specification matching the GitLab Merge Requests API	-
source_project_tag	No	Tag in the source project from which the Merge Request will be created. If not specified, the source project's branch is used	-
source_project_branch	No	Branch in the source project from which the Merge Request will be created	main
additionalIgnoreFiles	No	List of files containing paths to exclude from the MR. Populated similarly to .templateignore	-
values	No	Variables used during templating, in `key: value` format	-

How it works

The portal:

1. Clones the template repository used to generate the MR, based on its identifier (`source_project_id`). For more information, see Implementation details.
2. Reads the `values.yaml` file stored at the root of the repository and determines the default templating variables.
3. Reads the variables passed when the action is launched and merges them with the variables from `values.yaml`. Variables passed at launch take priority.
4. Reads the `.templateignore` file and determines the directories and files excluded from templating.
5. Renders the files from the templates, taking into account `values.yaml` and the variables passed to the action.
6. Changes the repository remote to the target repository identified by `target_project_id`, then runs `git push` to the branch in the source project (`source_project_branch`) or to the `main` branch.
7. Creates an MR according to the specified settings by sending a POST request to the GitLab API.

Note

The action performs a POST request to the URL: `/api/v4/projects/:id/merge_requests`.

CreateGitlabGroupVariables

> This action requires a token for the user on whose behalf it will run.

CreateGitlabGroupVariables — creates group-level variables in GitLab.

Request example

```
group_id: '0'
variables:
  - key: EXAMPLE_VARIABLE
    value: value
```

Request specification

Name	Required	Description
group_id	Yes	Identifier of the group in which to create the variables
variables	Yes	List of variables to create

The fields for each variable correspond to the official GitLab group-level variables API, `/groups/:id/variables`. For details, see the GitLab documentation.

Note

The action performs a POST request to the URL: `/api/v4/groups/:id/variables`.

CreateGitlabProjectVariables

> This action requires a token for the user on whose behalf it will run.

CreateGitlabProjectVariables — creates project-level variables in GitLab.

Request example

```
project_id: '0'
variables:
  - key: EXAMPLE_VARIABLE
    value: value
```

Request specification

Name	Required	Description
project_id	Yes	Identifier of the project in which to create the variables
variables	Yes	List of variables to create

The fields for each variable correspond to the official GitLab project-level CI/CD variables API, `/projects/:id/variables`. For details, see the GitLab documentation.

Note

The action performs a POST request to the URL: `/api/v4/projects/:id/variables`.

StartGitlabPipeline

> This action requires a token for the user on whose behalf it will run.

StartGitlabPipeline — starts a pipeline in GitLab.

Request example

```
project_id: 0
ref: main
variables:
  - key: example-key
    value: example-value
```

Request specification

Name	Required	Description
project_id	Yes	Identifier of the project in which to run the pipeline
ref	Yes	Name of the branch or tag, or the commit SHA, on which the pipeline will run
variables	No	List of variables to pass to the pipeline being run
variables.key	Yes	Variable name
variables.value	Yes	Variable value

Note

The action performs a POST request to the URL: `/api/v4/projects/:id/pipeline`.

DeleteGitlabProject

> This action requires a token for the user on whose behalf it will run.

DeleteGitlabProject — deletes an existing project in GitLab through the GitLab API. Authentication uses a GitLab token provided in the credentials.

Request example

```
project_id: 0
```

Request specification

Name	Required	Description
project_id	Yes	Identifier of the project to delete

Note

The action performs a DELETE request to the URL: `/api/v4/projects/:id`.

SonarQube

CreateSonarqubeProject

> Running this action requires a SonarQube User Token generated by the user on whose behalf the action will run.

CreateSonarqubeProject — creates a new project in SonarQube. The action uses the SonarQube Web API to create a project with the specified parameters, such as the project key, project name, main branch, new code definition parameters, and project visibility. Authentication is performed using a SonarQube token, which must be provided in the credentials.

Request example

```
project: example-project
name: example-project
mainBranch: develop
newCodeDefinitionType: NUMBER_OF_DAYS
newCodeDefinitionValue: '30'
visibility: public
```

Request specification

Name	Required	Description	Possible values	Default value
project	Yes	Unique project identifier (Project Key) in SonarQube	-	-
name	Yes	Project name displayed in the SonarQube interface	-	-
mainBranch	No	Name of the project's main branch	-	master
newCodeDefinitionType	No	Method for defining "new code"	PREVIOUS_VERSION, NUMBER_OF_DAYS, REFERENCE_BRANCH	-
newCodeDefinitionValue	No	Value for the "new code" definition (e.g., number of days, if the type is NUMBER_OF_DAYS)	-	-
visibility	No	Project visibility	private, public	private

DeleteSonarqubeProject

> Running this action requires a SonarQube User Token generated by the user on whose behalf the action will run.

DeleteSonarqubeProject — deletes a project from SonarQube.

Request example

```
project: example-project
```

Request specification

Name	Required	Description	Default value
project	Yes	Unique identifier (Project Key) of the project to delete	-

DeleteSonarqubeProjects

> Running this action requires a SonarQube User Token generated by the user on whose behalf the action will run.

DeleteSonarqubeProjects — deletes one or more projects from SonarQube.

Request example

```
analyzedBefore: 2017-10-19T13:00:00+0200
onProvisionedOnly: 'false'
projects: example_project,another_project
q: example
qualifiers: TRK
```

Request specification

Name	Required	Description	Possible values	Example values
analyzedBefore	No	Delete all projects whose last analysis is older than a specific date	-	2017-10-19, 2017-10-19T13:00:00+0200
onProvisionedOnly	No	Filter projects by the `onProvisionedOnly` value	true, false, yes, no	-
projects	No	List of project keys to delete	-	my_project, another_project
q	No	Delete all projects whose name or key contains the specified substring	-	example
qualifiers	No	Filter projects by the specified qualifiers	TRK, VW, APP	

Apache Kafka

CreateKafkaTopics

This action requires the following credentials:

- `user` — the username under which the action runs.
- `password` — the password for that user.

CreateKafkaTopics — creates new topics in Kafka.

Request example

```
securityProtocol: SASL_PLAINTEXT
saslMechanism: PLAIN
partitions: 1
replication_factor: 1
configs: {}
topics:
  - example_1
  - example_2
```

Request specification

Name	Required	Description	Possible values
securityProtocol	Yes	Protocol for connecting to Kafka. For more information, see the Kafka documentation	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL
saslMechanism	No	SASL authentication mechanism. Required when using the SASL_PLAINTEXT or SASL_SSL protocol. For more information, see the Kafka documentation	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512
partitions	Yes	Number of partitions into which each topic will be divided	-
replication_factor	Yes	Number of copies (replicas) of each topic partition to place on different brokers	-
configs	Yes	Key-value configuration for the topics to create	Values are listed in the Kafka documentation
topics	Yes	List of topic names to create	-

DeleteKafkaTopics

> This action requires the following credentials:
> - `user` — the username under which the action runs.
> - `password` — the password for that user.

DeleteKafkaTopics — deletes existing topics in Kafka.

Request example

```
securityProtocol: SASL_PLAINTEXT
saslMechanism: PLAIN
topics:
  - example_1
  - example_2
```

Request specification

Name	Required	Description	Possible values
securityProtocol	Yes	Protocol for connecting to Kafka. For more information, see the Kafka documentation	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL
saslMechanism	No	SASL authentication mechanism. Required when using the SASL_PLAINTEXT or SASL_SSL protocol. For more information, see the Kafka documentation	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512
topics	Yes	List of topic names to delete	-

CreateKafkaUsers

> This action requires the following credentials:
> - `user` — the username under which the action runs.
> - `password` — the password for that user.

CreateKafkaUsers — creates new SASL/SCRAM users in Kafka.

Request example

```
securityProtocol: SASL_PLAINTEXT
saslMechanism: PLAIN
users:
  - user: example_user_1
    password: example_password_user_1
    mechanism: SCRAM-SHA-256
    iterations: 4096
  - user: example_user_2
    password: example_password_user_2
    mechanism: SCRAM-SHA-256
    iterations: 4096
```

Request specification

Name	Required	Description	Possible values
securityProtocol	Yes	Protocol for connecting to Kafka. For more information, see the Kafka documentation	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL
saslMechanism	No	SASL authentication mechanism. Required when using the SASL_PLAINTEXT or SASL_SSL protocol. For more information, see the Kafka documentation	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512
users	Yes	Set of users to create	-
users.user	Yes	Username to create	-
users.password	Yes	Password for the user	-
users.mechanism	Yes	Authentication mechanism for the user	SCRAM-SHA-256, SCRAM-SHA-512
users.iterations	Yes	Number of iterations used to hash the password	From 4096 to 16384

DeleteKafkaUsers

This action requires the following credentials:
- `user` — the username under which the action runs.
- `password` — the password for that user.

DeleteKafkaUsers — deletes existing SASL/SCRAM users in Kafka.

Request example

```
securityProtocol: SASL_PLAINTEXT
saslMechanism: PLAIN
users:
  - user: example_user_1
    mechanism: SCRAM-SHA-256
  - user: example_user_2
    mechanism: SCRAM-SHA-256
```

Request specification

Name	Required	Description	Possible values
securityProtocol	Yes	Protocol for connecting to Kafka. For more information, see the Kafka documentation	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL
saslMechanism	No	SASL authentication mechanism. Required when using the SASL_PLAINTEXT or SASL_SSL protocol. For more information, see the Kafka documentation	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512
users	Yes	Set of users to delete	-
users.user	Yes	Username to delete	-
users.mechanism	Yes	Authentication mechanism associated with the user to delete	SCRAM-SHA-256, SCRAM-SHA-512

CreateKafkaACLs

This action requires the following credentials:

- `user` — the username under which the action runs.
- `password` — the password for that user.

CreateKafkaACLs — creates a set of ACLs in Kafka.

Request example

```
securityProtocol: SASL_PLAINTEXT
saslMechanism: PLAIN
acls:
  - topics:
      - example_1
    allow:
      - User:principal_2
      - Group:principal_3
    deny:
      - User:principal_4
      - Group:principal_5
    ops:
      - CREATE
      - READ
      - WRITE
      - DELETE
      - DESCRIBE
      - DESCRIBE_CONFIGS
      - ALTER
    pattern: LITERAL
  - topics:
      - example_6
    allow:
      - User:principal_7
    allow_hosts:
      - 127.0.0.1
    deny:
      - User:principal_8
    deny_hosts:
      - 127.0.0.1
    ops:
      - CREATE
    pattern: LITERAL
```

Request specification

Name	Required	Description	Possible values	Default value
securityProtocol	Yes	Protocol for connecting to Kafka. For more information, see the Kafka documentation	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL	-
saslMechanism	No	SASL authentication mechanism. Required when using the SASL_PLAINTEXT or SASL_SSL protocol. For more information, see the Kafka documentation	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512	-
acls	Yes	Set of ACLs to create	-	-
acls.ops	Yes	List of operations to which the rule applies	List of possible operations	-
acls.pattern	Yes	Pattern type	List of possible patterns	-
acls.topics	No	List of topic names to which the rule applies	-	-
acls.groups	No	List of group names to which the rule applies	-	-
acls.transactional_ids	No	List of transaction IDs to which the rule applies	-	-
acls.tokens	No		-	-

Name	Required	Description	Possible values	Default value
		List of tokens to which the rule applies		
acls.allow	No	List of principals (users or groups) allowed by the rule	-	-
acls.deny	No	List of principals (users or groups) denied by the rule	-	-
acls.hosts	No	List of hosts for which the operation is allowed	-	-
acls.deny_hosts	No	List of hosts for which the operation is denied	-	-

List of possible patterns

- ANY.
- MATCH.
- LITERAL.
- PREFIXED.

List of possible operations

For more information, see the Kafka documentation.

Topics:

- ALL.
- ALTER.
- ALTER_CONFIGS.
- CREATE.
- DELETE.
- DESCRIBE.
- DESCRIBE_CONFIGS.
- READ.

- WRITE.

Group:

- ALL.
- DELETE.
- DESCRIBE.
- READ.

TransactionalID:

- ALL.
- DESCRIBE.
- WRITE.

Tokens:

- DESCRIBE.

DeleteKafkaACLs

> This action requires the following credentials:
> - `user` — the username under which the action runs.
> - `password` — the password for that user.

DeleteKafkaACLs — deletes a set of ACLs in Kafka.

Request example

```
securityProtocol: SASL_PLAINTEXT
saslMechanism: PLAIN
acls:
  - topics:
      - example_1
    allow:
      - User:principal_2
      - Group:principal_3
    allow_hosts:
      - 127.0.0.1
    deny:
      - User:principal_4
      - Group:principal_5
    deny_hosts:
      - 127.0.0.1
    ops:
      - CREATE
      - READ
      - WRITE
      - DELETE
      - DESCRIBE
      - DESCRIBE_CONFIGS
      - ALTER
    pattern: LITERAL
  - any_topic: true
    any_group: true
    any_transactional_id: true
    any_allow: true
    any_allow_hosts: true
    any_deny: true
    any_deny_hosts: true
    ops:
      - ANY
    pattern: ANY
  - any_resource: true
    any_allow: true
    any_allow_hosts: true
    any_deny: true
    any_deny_hosts: true
    ops:
      - ANY
    pattern: ANY
```

Request specification

Name	Required	Description	Possible values	Default value
securityProtocol	Yes	Protocol for connecting to Kafka. For more information, see the Kafka documentation	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL	-
saslMechanism	No	SASL authentication mechanism. Required when using the SASL_PLAINTEXT or SASL_SSL protocol. For more information, see the Kafka documentation	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512	-
acls	Yes	Set of ACLs to delete	-	-
acls.ops	Yes	List of operations to which the rule to delete applies	List of possible operations	-
acls.pattern	Yes	Pattern type	List of possible patterns	-
acls.any_resource	No	All resources	-	false
acls.topics	No	List of topic names to which the rule applies	-	-
acls.any_topic	No	All topics	-	false
acls.groups	No	List of group names to which the rule applies	-	-
acls.any_group	No	All groups	-	false

Name	Required	Description	Possible values	Default value
acls.transactional_ids	No	List of transaction IDs to which the rule applies	-	-
acls.any_transactional_id	No	Any transaction	-	false
acls.tokens	No	List of tokens to which the rule applies	-	-
acls.any_token	No	All tokens	-	false
acls.allow	No	List of principals (users or groups) allowed by the rule	-	-
acls.any_allow	No	Any principal (user or group)	-	false
acls.allow_hosts	No	List of hosts for which the operation is allowed	-	-
acls.any_allow_hosts	No	Any host from which the operation is allowed	-	false
acls.deny	No	List of principals (users or groups) denied by the rule	-	-
acls.any_deny	No	Any principal (user or group)	-	false
acls.deny_hosts	No	List of hosts from which the operation is denied	-	-

Name	Required	Description	Possible values	Default value
acls.any_deny_hosts	No	Any host from which the operation is denied	-	false

List of possible patterns

- ANY.
- MATCH.
- LITERAL.
- PREFIXED.

List of possible operations

For a detailed description, see the Kafka documentation.

Topics:

- ALL.
- ALTER.
- ALTER_CONFIGS.
- CREATE.
- DELETE.
- DESCRIBE.
- DESCRIBE_CONFIGS.
- READ.
- WRITE.

Group:

- ALL.
- DELETE.
- DESCRIBE.
- READ.

TransactionalID:

- ALL.
- DESCRIBE.
- WRITE.

Token:

- DESCRIBE.

HashiCorp Vault

CreateVaultSecret

> Running this action requires a Vault token with permission to create secrets.

CreateVaultSecret — creates a secret with one or more values in HashiCorp Vault.

Request example

```
path: example/data/path
secrets:
  - key: key1
    value: value1
  - key: key2
    value: value2
```

Request specification

Name	Required	Description	Possible values	Default value
path	Yes	Path at which the secret will be stored in Vault		-
allow_update	No	Determines the action's behavior when creating or updating a secret	true, false, merge, merge_or_create	false
secrets	Yes	Set of secrets to create in Vault		-
secrets.key	Yes	Name (identifier) of the secret		-
secrets.value	Yes	Value of the secret		-

Note

`allow_update` determines the action's behavior when creating or updating a secret:

- `false` (default) — the action fails with an error if the secret already exists;
- `true` — a new version of the secret is created with the values passed to the action;

- `merge` — only the secret keys specified in the action are updated or created. Existing keys not mentioned in the action are preserved unchanged. If the secret does not yet exist at the path, the action fails with an error;
- `merge_or_create` — behaves like `merge` if the secret already exists; if the secret does not exist at the path, it is created (a full write of the values from the action).

DeleteVaultSecret

> Running this action requires a Vault token with permission to create secrets.

DeleteVaultSecret — deletes a secret from HashiCorp Vault.

Request example

```
path: example/data/path
```

Request specification

Name	Required	Description
path	Yes	Path at which the secret to be deleted is located in Vault

CreateVaultKubernetesAuthRole

> Running this action requires a Vault token with permission to create or update Kubernetes auth backend roles.

CreateVaultKubernetesAuthRole — creates or updates a Kubernetes authentication role in HashiCorp Vault.

Request example

```
mountPath: kubernetes
role: example
bound_service_account_names:
  - default
bound_service_account_namespaces:
  - default
optional:
  token_ttl: 1h
  token_max_ttl: 12h
  audience: vault
  token_policies:
    - default
```

Request specification

Name	Required	Description
mountPath	Yes	Mount path of the Kubernetes auth backend in Vault (e.g., kubernetes)
role	Yes	Name of the role to create in Vault
bound_service_account_names	Yes	List of service account names allowed to access via this role
bound_service_account_namespaces	Yes	List of namespaces allowed to access via this role
optional	No	Additional role parameters (listed in the table below)

Supported values for `optional`:

Field	Type	Description
token_ttl	string	Time-to-live (TTL) of the token issued at login
token_max_ttl	string	Maximum TTL of the token
token_policies	[]string	Additional policies assigned at login
audience	string	Value of the JWT audience (aud) that Vault expects from the token
token_period	string	Token renewal period
token_explicit_max_ttl	string	Explicit upper bound on the token's TTL
token_num_uses	int	Limit on the number of times the token can be used
token_type	string	Type of token issued (e.g., service, batch)
alias_name_source	string	Source of the alias name for identity
token_no_default_policy	bool	Exclude the default policy from the token
token_bound_cidrs	[]string	Restriction on the CIDR ranges from which the issued token can be used

The full list of supported parameters is provided in the official HashiCorp Vault documentation.

Kubernetes

CreateKubernetesResource

> This action requires a Kubernetes service account token.

CreateKubernetesResource — creates one or more resources in a Kubernetes cluster or updates existing resources.

Request example

```
manifests:
  - apiVersion: v1
    kind: Namespace
    metadata:
      name: example1
  - apiVersion: v1
    kind: Namespace
    metadata:
      name: example2
```

Request specification

Name	Required	Description
manifests	Yes	Kubernetes manifests to apply

GetKubernetesResource

> This action requires a Kubernetes service account token.

GetKubernetesResource — retrieves a resource from a Kubernetes cluster.

Request example

```
group: managed-services.deckhouse.io
version: v1alpha1
resource_type: postgres
resource_name: example-postgres
namespace: default
```

Request specification

Name	Required	Description	Possible values
group	Yes	API group of the resource. Specifies the API group to which the requested object belongs	Determining the required Group and Version
version	Yes	API version of the resource	Determining the required Group and Version
resource_type	Yes	Type of resource to retrieve	-
resource_name	Yes	Name of the resource to retrieve	-
namespace	Yes	Namespace containing the resource	-

Response

On success, the action returns the resource object in the `resource` field. If the resource is not found, the action fails with an error.

Name	Description
`resource`	Resource object in Kubernetes format

Determining the required Group and Version

Each resource type has its own API group (Group) and version (Version). A complete list of API resources, groups, and versions is available in the Kubernetes documentation.

If you do not know which API groups and versions are required, you can look up the current values. There are several ways to determine them:

Using the `d8 k` utility

The `d8 k explain` command shows the `apiVersion` for a resource.

Example:

```
d8 k explain deployment
```

Example output:

```
GROUP:      apps
KIND:       Deployment
VERSION:    v1

DESCRIPTION:
    Deployment enables declarative updates for Pods and ReplicaSets.

FIELDS:
...
```

Using the documentation

1. Find the resource you need (for example, Deployment).
2. Find the API group and version in the header. For example, for Deployment:

   ```
   apiVersion: apps/v1
   ```

   Here:

   - `API Group` is `apps`;
   - `Version` is `v1`.

DeleteKubernetesResource

> This action requires a Kubernetes service account token.

DeleteKubernetesResource — deletes an existing resource in a Kubernetes cluster.

Request example

```
group: apps
version: v1
resource_type: deployments
resource_name: nginx-deployment
namespace: example
```

Request specification

Name	Required	Description	Possible values
group	Yes	API group of the resource. Specifies the API group to which the object being deleted belongs	Determining the required Group and Version
version	Yes	API version of the resource	Determining the required Group and Version
resource_type	Yes	Type of resource to delete	pods, services, deployments, statefulsets, daemonsets, replicasets, jobs, cronjobs, nodes, namespaces, configmaps, secrets, persistentvolumes, persistentvolumeclaims, limitranges, resourcequotas, horizontalpodautoscalers, ingresses, networkpolicies, serviceaccounts, roles, clusterroles, rolebindings, clusterrolebindings, podsecuritypolicies, storageclasses, volumeattachments, events, endpoints, customresourcedefinitions
resource_name	Yes	Name of the resource to delete	-
namespace	Yes		-

Name	Required	Description	Possible values
		Namespace containing the resource	

Keycloak

CreateKeycloakClient

> This action requires the following credentials:
> - `username` — the username under which the action runs.
> - `password` — the password for that user.

CreateKeycloakClient — creates a new client in Keycloak.

Request example

```
realm: master
config:
  clientId: example
  name: example
  enabled: true
  clientAuthenticatorType: client-secret
  secret: secret
  defaultClientScopes:
  - roles
  - profile
  - email
  optionalClientScopes:
  - address
  - phone
  - offline_access
```

Request specification

Name	Required	Description	Possible values
realm	Yes	Keycloak realm in which to create the client	-
config	Yes	Parameters for the client to create, as defined in the Keycloak ClientRepresentation specification	-

Nexus Repository

CreateNexusRepository

> Running this action requires a token — a base64(`admin:password`) string used for HTTP Basic authentication in requests to Nexus.

`CreateNexusRepository` — creates a new repository of any supported type (maven, docker, npm, etc.) in Nexus Repository Manager 3 using the REST API.
The format, type, and other key settings are fully configurable and correspond to the Nexus API.

Request example (Maven hosted)

```
description: |
  Maven hosted repo for internal Java build artifacts.
name: my-maven-repo
format: maven
type: hosted
online: true
storage:
  blobStoreName: default
  strictContentTypeValidation: true
  writePolicy: ALLOW
cleanup:
  policyNames:
    - maven-cleanup
maven:
  versionPolicy: RELEASE
  layoutPolicy: PERMISSIVE
```

Request example (Docker group)

```
description: |
  Docker group repo aggregating hosted+proxy.
name: my-docker-group
format: docker
type: group
online: true
storage:
  blobStoreName: default
  strictContentTypeValidation: true
group:
  memberNames:
    - docker-hosted
    - docker-proxy
docker:
  v1Enabled: false
  forceBasicAuth: true
  httpPort: 5001
```

Request specification

Field	Required	Description	Example
description	No	Description of the action or repository's purpose. Used only in the UI, not by Nexus itself	-
name	Yes	Name of the repository to create. Must be unique within Nexus	my-maven-repo
format	Yes	Format (`maven` , `docker` , `npm` , `raw` , etc.)	maven
type	Yes	Type: `hosted` , `proxy` , or `group`	hosted
online	Yes	Whether the repository is available (`true` / `false`)	true
storage	Yes	Storage object: `blobStoreName` , `strictContentTypeValidation` , `writePolicy`	Example
cleanup	No	Linked cleanup policies (`policyNames`)	policyNames: [maven-cleanup]
maven	For maven	Maven-only: `versionPolicy` , `layoutPolicy`	Example
proxy	For proxy	Proxy repository: `remoteUrl` , `contentMaxAge` , `metadataMaxAge`	-
group	For group	List of `memberNames` values	Example
docker	For docker		Example

Field	Required	Description	Example
		Docker-specific parameters: `httpPort`, `v1Enabled`, `forceBasicAuth`	
component	Very rare	Only for certain non-standard scenarios	-
attributes	No	Any custom fields	-

Requirements

- Use only the blocks (`maven`, `group`, `proxy`, `docker`, etc.) supported by the selected type and format.
- For Maven hosted repositories, `maven: {versionPolicy, layoutPolicy}` is required.
- For group repositories, `group.memberNames` is required.
- For proxy repositories, `proxy.remoteUrl` is required.
- For Docker repositories, specify the Docker-specific fields in `docker`.

DeleteNexusRepository

> Running this action requires a token — a base64(`admin:password`) string used for HTTP Basic authentication in requests to Nexus.

DeleteNexusRepository — deletes an existing repository from Nexus Repository Manager 3.

Request example

```
name: my-repo-to-delete
```

Request specification

Field	Required	Description
`name`	Yes	Name of the repository to delete

Algorithm

- Sends a `DELETE` request to `/service/rest/v1/repositories/{name}`, where `{name}` is the value of the `name` field.
- If the repository is found and deleted, the request returns status code 204.

- If the repository is not found, the request returns a 404 error.

CreateNexusPrivilege

> Running this action requires a token — a base64(`admin:password`) string used for HTTP Basic authentication in requests to Nexus.

CreateNexusPrivilege — creates a new privilege in Nexus Repository Manager 3. Privileges define access rights to repositories and other Nexus resources.

Request example (repository-view)

```
name: example-privilege
description: Example privilege description
type: repository-view
actions:
  - READ
  - BROWSE
format: maven2
repository: maven-releases
```

Request example (repository-content-selector)

```
name: content-selector-privilege
description: Privilege with content selector
type: repository-content-selector
actions:
  - READ
format: maven2
repository: maven-releases
contentSelector: my-content-selector
```

Request example (wildcard)

```
name: wildcard-privilege
description: Wildcard privilege
type: wildcard
pattern: nx-*
actions:
  - READ
```

Request specification

Field	Required	Description
name	Yes	Name of the privilege to create. Must be unique within Nexus
description	No	Privilege description
type	Yes	Privilege type: `repository-view`, `repository-content-selector`, `repository-admin`, `application`, `wildcard`
actions	No	List of actions allowed by the privilege (e.g., `READ`, `BROWSE`, `CREATE`, `UPDATE`, `DELETE`)
format	No	Repository format (e.g., `maven2`, `docker`, `npm`). Used for the `repository-view`, `repository-content-selector`, `repository-admin` types
repository	No	Repository name. Used for the `repository-view`, `repository-content-selector`, `repository-admin` types
contentSelector	No	Content selector name. Required for the `repository-content-selector` type. If not specified or invalid, the type is automatically converted to `repository-view`
pattern	No	Pattern for the `wildcard` type
domain	No	Domain for the `application` type
attributes	No	Additional key-value parameters

Note

For the `repository-content-selector` type, the content selector must exist in Nexus before the privilege is created. If the content selector is not specified or is invalid, the action automatically converts the privilege type to `repository-view`.

AssignNexusPrivilege

> Running this action requires a token — a base64(`admin:password`) string used for HTTP Basic authentication in requests to Nexus.

AssignNexusPrivilege — assigns privileges to an existing role in Nexus Repository Manager 3. The action retrieves the role's current configuration and merges its existing privileges with the new ones.

Request example

```
roleId: example-role
privileges:
  - example-privilege
  - another-privilege
```

Request specification

Field	Required	Description
roleId	Yes	Identifier of the role to which privileges are assigned
privileges	Yes	List of privilege names to assign to the role

How it works

1. Retrieves the role's current configuration from Nexus.
2. Merges the role's existing privileges with the new privileges from the request.
3. Updates the role with the merged list of privileges.

Note

The role must exist in Nexus before privileges can be assigned. If the role is not found, the action fails with an error. All specified privileges must also exist in Nexus.

DeleteNexusPrivilege

> Running this action requires a token — a base64(`admin:password`) string used for HTTP Basic authentication in requests to Nexus.

DeleteNexusPrivilege — deletes a privilege from Nexus Repository Manager 3.

Request example

```
name: example-privilege
```

Request specification

Field	Required	Description
name	Yes	Name of the privilege to delete

CreateNexusRole

> Running this action requires a token — a base64(`admin:password`) string used for HTTP Basic authentication in requests to Nexus.

CreateNexusRole — creates a new role in Nexus Repository Manager 3. Roles combine privileges and can include other roles.

Request example

```
id: example-role
name: Example Role
description: Example role description
privileges:
  - nx-repository-view-*-*-read
  - nx-repository-view-maven2-*-browse
roles: []
```

Request specification

Field	Required	Description
id	Yes	Unique role identifier
name	Yes	Role name
description	No	Role description
privileges	No	List of privilege names assigned to the role
roles	No	List of other role identifiers included in this role

AssignNexusRole

> Running this action requires a token — a base64(`admin:password`) string used for HTTP Basic authentication in requests to Nexus.

AssignNexusRole — assigns roles to an existing user in Nexus Repository Manager 3. The action retrieves the user's current configuration and merges the user's existing roles with the new ones.

Request example

```
userId: example-user
roles:
  - example-role
  - another-role
```

Request specification

Field	Required	Description
userId	Yes	Identifier of the user to which roles are assigned
roles	Yes	List of role identifiers to assign to the user

How it works

1. Retrieves the user's current configuration from Nexus.
2. Merges the user's existing roles with the new roles from the request.
3. Updates the user with the merged list of roles.

Note

The user must exist in Nexus before roles can be assigned. If the user is not found, the action fails with an error. All specified roles must also exist in Nexus.

DeleteNexusRole

> Running this action requires a token — a base64(`admin:password`) string used for HTTP Basic authentication in requests to Nexus.

DeleteNexusRole — deletes a role from Nexus Repository Manager 3.

Request example

```
id: example-role
```

Request specification

Field	Required	Description
id	Yes	Identifier of the role to delete

CreateNexusUser

> Running this action requires a token — a base64(`admin:password`) string used for HTTP Basic authentication in requests to Nexus.

CreateNexusUser — creates a new user in Nexus Repository Manager 3.

Request example

```
userId: example-user
firstName: First
lastName: Last
emailAddress: user@example.com
password: password
status: active
roles:
  - nx-admin
```

Request specification

Field	Required	Description
userId	Yes	Unique user identifier
firstName	Yes	User's first name
lastName	Yes	User's last name
emailAddress	Yes	User's email address
password	Yes	User's password
status	Yes	User status: `active` or `disabled`
roles	No	List of role identifiers assigned to the user at creation

DeleteNexusUser

> Running this action requires a token — a base64(`admin:password`) string used for HTTP Basic authentication in requests to Nexus.

DeleteNexusUser — deletes a user from Nexus Repository Manager 3.

Request example

```
userId: example-user
```

Request specification

Field	Required	Description
userId	Yes	Identifier of the user to delete

DefectDojo

CreateDefectdojoProduct

> This action requires an API v2 key for the user on whose behalf it will run.

CreateDefectdojoProduct — creates a new product in DefectDojo. The action uses the DefectDojo API v2.

Request example

```
name: example
description: example description
prod_type: 1
```

Request specification

The fields correspond to the official DefectDojo API endpoint `/api/v2/products`. For details, see the DefectDojo documentation.

DeleteDefectdojoProduct

> This action requires an API v2 key for the user on whose behalf it will run.

DeleteDefectdojoProduct — deletes a product from DefectDojo. The action uses the DefectDojo API v2.

Request example

```
id: 1
```

Request specification

Name	Required	Description	Default value
id	Yes	Identifier of the product to delete	-

CreateDefectdojoEngagement

> This action requires an API v2 key for the user on whose behalf it will run.

CreateDefectdojoEngagement — creates a new engagement in DefectDojo. The action uses the DefectDojo API v2.

Request example

```
name: example engagement
product: '1'
target_start: '2024-06-01'
target_end: '2024-06-30'
lead: '1'
```

Request specification

The fields correspond to the official DefectDojo API endpoint `/api/v2/engagements`. For details, see the DefectDojo documentation.

CodeScoring

CreateCodeScoringProject

CreateCodeScoringProject — creates a new project in CodeScoring. The action uses the CodeScoring API to register a project with the specified parameters:

- project name,
- repository URL,
- VCS ID,
- an option to automatically run SCA analysis after cloning the repository.

Request example

```
name: example-project
repository: https://gitlab.example.com/group/project.git
vcs_id: 2
run_sca_after_clone: true
```

Request specification

Name	Required	Description
name	Yes	Project name in CodeScoring
repository	Yes	Repository URL (e.g., https://gitlab.example.com/group/project.git)
vcs_id	Yes	VCS ID in CodeScoring (must be greater than 0)
run_sca_after_clone	No	Automatically run SCA analysis after cloning the repository

Response

The response contains the created project object with the following information: project identifier (pk), name, project type, description, repository information, project status, access permissions, license, number of dependencies and vulnerabilities, project languages, scan schedule status, and the dates of the first and last SCA scans.

DeleteCodeScoringProject

DeleteCodeScoringProject — deletes a project in CodeScoring by its ID.

Request example

```
id: 1
```

Request specification

Name	Required	Description
id	Yes	Project ID in CodeScoring

Debug

Debug

Debug runs a debugging action. It performs a specified number of wait cycles, writes arbitrary data to the log, and returns that data in the action response.

Request example

```
sleep_time: 1
sleep_count: 3
extra:
  example_key: example_value
```

Request specification

Name	Required	Description	Default value
sleep_time	No	Duration of one wait cycle, in seconds	`1`
sleep_count	No	Number of wait cycles	`1`
extra	No	An arbitrary set of key-value pairs that is written to the log and returned in the action’s response	-

Fail

Fail simulates an action execution error. It is intended for use as a debugging step in processes.

Request example

```
fail: true
```

Request specification

Name	Required	Description	Default value
fail	No	If `true`, the action completes with an error; if `false`, the action completes successfully	`true`

Wait

Wait

Wait pauses execution for a specified number of seconds and can optionally add a random duration (jitter). It is intended for use in processes as a delay element, including while waiting for the results of a previous action to be applied.

Request example

```
duration_seconds: 10
max_jitter_seconds: 0
description: "Waiting for release"
```

Request specification

Name	Required	Description	Default value
duration_seconds	Yes	Base wait duration in seconds (0–86400, i.e. up to 24 hours)	-
max_jitter_seconds	No	Maximum random addition to the wait time in seconds (0–N). 0 disables jitter	`0`
description	No	Description shown in the logs and the action’s response	-

Processes

Overview

Processes are an automation mechanism for complex business scenarios that lets you build visual execution diagrams for actions, with support for conditional logic, parallel execution, and error handling.

Key concepts

Process elements

A process consists of various types of elements:

- “Start” — the process entry point.
- “Task” — runs a specific action.
- “Exclusive gateway” — conditional branching.
- “Parallel gateway” — merges several process branches into one.
- “Loop” — repeats part of the process a fixed number of times, or iterates over a JSON array from the store.
- “Template” — evaluates a Go template and writes the result to the process store.
- “Timer” — pauses until a specified point in time.
- “Note” — a text block.
- “Error” — immediately stops the process with the `Failed` status.
- “End” — completes the process.

Creating a process

Basic information

To create a process, go to “Self-service” → “Processes” and click “Create”.

Fill in the following fields:

- “Name” — the process name.
- “Description” — a detailed description of the process’s purpose.
- “Resource” — one or more resources for which the process can be run.
- “Owner” — the user responsible for the process.
- “Owner team” — the team responsible for the process.
- “Tags” — tags used to categorize the process.
- “Icon” — the icon displayed in the interface.

Process configuration

The process is configured in the visual editor, on the “Configuration” tab.

> To configure a process, you must first fill in the basic fields and create it. After that, the “Configuration” and “Parameters” tabs become available.

Adding elements

To add elements to the diagram:

1. Add an element by selecting its type in the editor panel.
2. Configure the element’s parameters in the side panel (action, conditions, etc.).
3. Connect the elements to each other.

Element types

Task

Each task runs a specific action previously created in “Self-service” → “Actions”.

Exclusive gateway

An exclusive gateway lets you configure process branching based on conditions. By default, the status of the previous task is checked, and depending on it, execution follows the “success” or “failure” branch.

As conditions for an exclusive gateway, you can set either a check of the previous task’s status, or a comparison of values from templates:

- To check the status of the previous task, use the construct `{{ .prev_task.status }}`.
- To check a value from the store, use `{{ .store.<path> }}`, including nested keys: `{{ .store.notification.status }}`.
- Inside a loop — fields of the current item: `{{ .store._loop.item.scan_type }}`, and the flags `{{ .store._loop.first }}` and `{{ .store._loop.last }}`.

You can check multiple conditions and combine their results using the `AND` or `OR` operator.

If the condition check passes, the process follows the “True” branch (green); if not, it follows the “False” branch (red). The “True” and “False” ports can be placed on any side of the gateway; a connection can be moved to a different port.

Parallel gateway

Used to merge several branches into one, based on specified conditions.

In the parallel gateway’s configuration, you can set the following waiting parameters:

- Wait for all incoming elements to complete before continuing execution.
- Wait for at least one incoming element to complete before continuing execution.

You can also configure what exactly counts as “completion”:

- Only successful completion of the tasks feeding into the gateway.
- Any final status of the incoming tasks (`Failed`, `Skipped`, etc.).

A parallel gateway can also be used to split branches, if connected after “Exclusive gateway” or “Loop” elements, or as a helper element to improve the readability of the process diagram.

Loop

The “Loop” element repeats a branch of the process either a fixed number of times, or once for each element of a JSON array from the store. Once all iterations are complete, the element connected to the loop’s exit is activated.

Loop mode

Two modes are available:

- “Fixed number” — the “Number of iterations” field, from 1 to 10000. On each iteration, `{{ .store._loop.item }}` holds the iteration number (1, 2, 3…).
- “By collection” — the “Collection template” field: a Go template that, when entering the loop, must produce a JSON array. Typically, this is a path to an array written by a previous task: `{{ .store.notification.engagements }}`. Each array element is one iteration; the current element is available as `{{ .store._loop.item }}`.

If the array is empty or the key is missing, the loop body is not executed, and the process immediately follows the “Loop exit” connection.

For more information, see Process store.

Outgoing connections

Exactly two connections to different elements must originate from the “Loop” element:

1. “Loop body” — the branch built from this port is executed on each iteration.
2. “Loop exit” — execution continues along this branch after the last iteration.

The “Loop body” and “Loop exit” ports can be placed on any side of the element; a connection can be moved to a different port.

Usage

Typical scenarios:

- repeatedly checking the status of an external system with a fixed number of attempts;
- iterating over a list of objects from an API response: a task writes the array to the store, and a loop in “By collection” mode processes each element;
- nested loops — the parent context is available via `{{ .store._loop.parent }}`.

Template

The “Template” element evaluates a Go template without calling an additional action, and writes the result to the store as a string.

Configure:

- “Template body” — text with placeholders `{{ .store.* }}`, `{{ .process.* }}`, and, inside a loop, `{{ .store._loop.* }}`;
- “Store key” — the destination dot-path, e.g. `rendered_message`;
- “Format hint” — how to display the value when viewing the run (`text`, `markdown`, `html`, `json`); does not affect the store’s content.

For examples and limitations, see Process store.

Timer

The “Timer” element pauses process execution until a specified point in time, after which the next element is activated.

While the process is only waiting for the timer to fire (no other active tasks), the run is placed in the “Wait” status. The run visualization panel shows a banner with the estimated resumption time.

Outgoing connections

Exactly one connection to the next element must originate from the “Timer” element. Connection is only possible through the left (input) and right (output) ports.

If the timer is passed through again within the same run (for example, via a loop), the trigger time is recalculated.

Schedule modes

In the timer’s configuration, select a “Schedule”:

- “Delay after entering the element” — a fixed pause from the moment the process reaches the timer. Set the “Delay (seconds)” from 1 to 1,209,600 (14 days).
- “Custom schedule” — triggers at a specified calendar time in the selected time zone (IANA, e.g. `Europe/Moscow`).

For “Custom schedule” mode, specify a “Pattern”:

- “Specific day of the week” — “Day of the week”, “Time of day” (hour and minute), “Time zone”.
- “Specific day of each month” — “Day of month” (1–31), “Time of day”, “Time zone”. If the month does not have that day, the last day of the month is used.
- “Every N days” — “Every N days” (1–365), “Time of day”, “Time zone”. The first trigger is no earlier than N calendar days from the day the process reached the timer; upon re-entering the element, the calculation is redone.

“Time of day” is set in the selected time zone (the “Time zone” field), if it differs from the browser’s time zone.

Usage

Typical scenarios: a pause before re-checking the status of an external system, a delayed start of the next step, waiting for a maintenance window, or a regular calendar slot.

Note

The “Note” element is intended for text annotations on the process diagram: it is not executed at runtime and is not connected to other elements.

In the note’s configuration, you can set:

- “Text” — content with Markdown support.
- “Background color” — the fill color.
- “Text color” — the text color.

A note is always positioned behind other elements and can be used to visually highlight parts of the process. It is also the only element whose size can be changed.

Error

The “Error” element forcibly transitions a running process to the `Failed` status: when this element is reached, execution of all process actions is interrupted.

No connections to subsequent elements should originate from this element.

Process parameters

The “Parameters” tab is used to configure process parameters that can be used in all actions within the process.

The configuration and use of process parameters are described in Templating.

Process store

The process store is a JSON object used to pass data between steps of a single run: identifiers, arrays, intermediate results, and text produced by the “Template” element.

For a full description of write rules, operations, loop context, and example scenarios, see Process store.

In brief:

- **One store per run** — each process instance has its own store; it can be viewed on the “Store” tab when visualizing a run.
- **Nested paths** — keys are specified with dots: `notification.module_name`, `ctx.job.id`; array indices are supported: `items[0].status`.
- **Rule-based writes** — data is written to the store after an action completes successfully (see Process store update), or when the “Template” element runs. Available operations include writing strings and JSON, appending, merging, and deleting, plus a condition for each rule.
- **Reading** — Go templates `{{ .store.<path> }}` in action configuration and gateway conditions (see Process store).

If a key is not present in the store, the template `{{ .store.<path> }}` will cause the step to fail with an error.

Running a process

Manual run

To run a process manually:

1. Go to the entity for which the process needs to be run.
2. In the entity's menu, select "Run process".
3. Choose the desired process from the list.
4. Fill in the process parameters.
5. Click "Run".

Launch parameters

The following are available when launching a process:

- "Common process parameters" — parameters defined in the process configuration.
- "Action parameters" — parameters for each action in the process.
- "Environment variables" — additional variables for execution.

Execution management

Process statuses

A process can be in one of the following statuses:

- "Created" — the process has been created but not run.
- "Running" — the process is currently running.
- "Paused" — process execution has been paused.
- "Completed" — the process completed successfully.
- "Failed" — the process finished with an error.
- "Cancelled" — process execution was cancelled.

Management operations

The following operations are available for active processes:

- "Pause" — temporarily stop execution.
- "Resume" — continue execution after pausing.
- "Stop" — completely stop execution.
- "Force restart" — restart the process from the beginning.

State tracking

In the "Process runs" section, you can view:

- A list of all process runs for the entity.

- Detailed information about each run.
- Action execution logs.
- The status of each process element.
- The process execution timeline.
- The contents of the process store on the “Store” tab.

Timeline

The run visualization panel has a “Timeline” tab that shows the process execution history: for each element that participated in the current run, the start and end times, duration, and status are shown.

Process log

A detailed execution log is available for each process run: a panel on the run diagram, a separate window, or a dock at the bottom of the screen. The log and the store’s JSON can be downloaded from the process run dialog panel.

Usage examples

Creating a configured project

Typical process diagram:

1. “Start” — starts the process.
2. “Task” — creates a project in GitLab.
3. “Exclusive gateway” — checks whether creation succeeded.
4. “Task” (on success) — configures the project’s variables.
5. “Task” (on failure) — sends an error notification.
6. “End” — completes the process.

Deploying an application

Typical process diagram:

1. “Start” — starts the deployment process.
2. “Parallel gateway” — splits into branches.
3. “Task” (branch 1) — creates a namespace in Kubernetes.
4. “Task” (branch 2) — creates secrets in Vault.
5. “Parallel gateway” — waits for both branches to complete.
6. “Task” — deploys the application.
7. “End” — completes the process.

Limitations

The following limitations apply to processes:

- Processes cannot contain more than 100 elements.
- The maximum process execution time is 24 hours.
- The number of concurrent process runs is limited by system settings.

- Some actions may not be available for use in processes.

Process store

The process store is a JSON object shared by a single process run. Tasks use it to pass data to each other: identifiers, arrays for loop iteration, intermediate results, and generated text.

Writing is done **only via rules** in the configuration of actions and “Template” elements. Reading is done via Go templates `{{ .store.<path> }}` in the request body, gateway conditions, and other fields that support templating.

Data model

- **One store per run** — each process instance has its own store; it is preserved between steps and available when viewing the run.
- **Nested paths** — keys are specified as dot-paths without a leading dot: `notification.module_name`, `ctx.job.id`. Intermediate objects are created automatically.
- **Array indices** — a path can reference an array element: `items[0].status`, `branches[2].name`.
- **Value types** — depending on the operation, strings, numbers, objects, and JSON arrays can end up in the store. The “Write string” and “Append string” operations always work with strings; operations with the JSON suffix store the parsed structure.

> The service key `_loop` is populated by the process engine while a loop is running. Do not write to `_loop` manually using action rules — use it for reading in templates only.

Store update rules

Rules are configured in the action: “Configuration” → “Update” → “Process store update”. They run **after the action completes successfully**, **in list order**. Each subsequent rule sees the changes made by the previous rules of the same action.

In the process task configuration, rules are shown in read-only mode; to change them, open the action from the task’s side panel.

Rule fields

Field	Required	Description
Condition	No	A Go template. An empty value means the rule always runs. After rendering, the result must be `true`, `false`, `1`, or `0`; otherwise the rule is skipped
Operation	Yes	The way the value at the “Target” path is changed
Target	Yes	Dot-path in the store, without a leading dot
Source	Yes*	Go template for the value. Not used for the “Delete” operation

Operations

Operation	Result
Write string	Replaces the value at the path with the text string from the source; JSON is not parsed
Write JSON	Replaces the value with the parsed JSON (object, array, number, boolean) from the source template
Append string	Appends text to the existing string at the path; if the key doesn’t exist, a string is created. Not applicable to numbers and arrays
Append JSON	Adds a single JSON item to the array at the path; if the array doesn’t exist, an empty array is created
Merge JSON (shallow)	Merges the JSON object from the source with the object at the path: top-level keys are overwritten, nested objects are replaced entirely
Delete	Deletes the key at the path

For the **Write JSON**, **Append JSON**, and **Merge JSON** operations, it is convenient to use the `toJSON` function together with a direct reference to a response field, e.g. `{{ .response.items }}` — the portal will substitute the value without extra serialization.

Rule examples

Save an identifier from the action’s response:

Condition	Operation	Target	Source
	Write string	`deploy.job_id`	`{{ .response.id }}`

Write an array for a subsequent loop:

Condition	Operation	Target	Source
	Write JSON	`notification.engagements`	`{{ toJSON .response.engagements }}`

Append a string to a log:

Condition	Operation	Target	Source
	Append string	`run.log`	`step {{ .store._loop.index }}: ok\n`

Update only part of an object:

Condition	Operation	Target	Source
	Merge JSON (shallow)	`meta`	`{{ toJSON (dict "status" "ready" "updated_by" .entity.slug) }}`

Run a rule only on the first loop iteration:

Condition	Operation	Target	Source
`{{ .store._loop.first }}`	Write string	`run.started_at`	`{{ now }}`

Delete a temporary key after use:

Condition	Operation	Target	Source
	Delete	`temp.payload`	

“Store paths” panel

The action’s rule editor has a **Store paths** panel: it shows the loop context keys (`_loop.item`, `_loop.index`, etc.).

Reading from the store

In process templates, use:

```
{{ .store.<path> }}
```

For nested fields, the path matches the “Target” in the write rules:

```
{{ .store.notification.module_name }}
{{ .store.items[0].id }}
```

Process launch parameters are available in a separate context:

```
{{ .process.<parameter_identifier> }}
```

For more information on templating contexts, see Templating.

> ⚠ If a key is not present in the store (the action has not yet run, the rule didn’t fire, or the condition was false), the template `{{ .store.<path> }}` will cause the step to fail with an error. Make sure preceding tasks have written the required data.

Loop context `_loop`

While the loop body is running, the `_loop` object is available in the store:

Key	Description
`_loop.item`	The current collection item (“By collection” mode) or the iteration number, starting from 1 (“Fixed number” mode)
`_loop.index`	The iteration index, starting from 0
`_loop.total`	The total number of iterations
`_loop.first`	`true` on the first iteration
`_loop.last`	`true` on the last iteration
`_loop.parent`	The parent loop’s context, for nested loops
`_loop.loop_element_uuid`	The UUID of the “Loop” element

Examples in templates:

```
{{ .store._loop.item.name }}
{{ .store._loop.item.scan_type }}
{{ .store._loop.index }}
```

In exclusive gateway conditions inside a loop:

```
{{ eq .store._loop.item.branch .store.current_branch_tag }}
```

Looping over a collection from the store

The “Loop” element supports two modes (the “Loop mode” field):

1. **Fixed number** — the body runs a specified number of times (1–10000). `_loop.item` holds the iteration number (1, 2, 3…).
2. **By collection** — when entering the loop, the “Collection template” Go template is evaluated; each element of the resulting **JSON array** is one iteration.

Collection template

The recommended approach is a path to an array already written to the store by a previous task:

```
{{ .store.notification.engagements }}
```

An alternative is to build the array in the template:

```
{{ toJSON .store.modules }}
```

Edge cases:

- An empty array or a missing key — 0 iterations, execution follows the “Loop exit” connection; a warning appears in the process log.
- A value at the path that is not a JSON array — a validation or loop execution error.

Scenario: iterating over a list from an API

1. **Task** “Get engagements” — a **Write JSON** rule, target `notification.engagements`, source `{{ toJSON .response.engagements }}`.
2. **Loop**, **By collection** mode, collection template `{{ .store.notification.engagements }}`.
3. **Task** in the loop body — use `{{ .store._loop.item.id }}`, `{{ .store._loop.item.name }}` in the request body.
4. **Exclusive gateway** in the body — a condition based on an item field, e.g. left value `{{ .store._loop.item.status }}`, right value `active`.
5. **Loop exit** connection — continuation after iterating over all elements.

The Template element

The “Template” element evaluates a Go template as the process runs and writes a **string** to the store. JSON in the result is not parsed — the text is stored in the store as-is.

Field	Description
Template body	A Go template; `{{ .store.* }}`, `{{ .process.* }}` are available, and, inside a loop, `{{ .store._loop.* }}`
Store key	The destination dot-path, e.g. `rendered_message` or `notification.summary`
Format hint	A hint used when viewing the run (`text`, `markdown`, `html`, `json`); does not affect the write

Example body:

```
## Report for {{ .store._loop.item.name }}

Status: {{ .store._loop.item.status }}
Owner: {{ .store._loop.item.owner }}
```

Subsequent tasks read the result via `{{ .store.rendered_message }}` or the corresponding path.

Viewing the store during execution

On the process run visualization screen, open the **Store** tab:

- **Tree** — navigate nested keys; the type and value of the selected node are shown on the right.
- **JSON** — the full contents of the store in JSON format.

You can copy a value, refresh the data, and download the run's JSON (button on the run dialog panel).

The task's side panel on the run diagram shows the selected action's **Process store update rules** — useful for comparing the store's actual contents against the configuration.

Process log and debugging

The visualization panel provides:

- an **Execution log** tab (including in a separate window or below the diagram);
- **Download log** — the run's text log;
- messages about an empty loop collection and skipped store rules.

If a template error occurs, or invalid JSON is used in a rule with a JSON operation, that rule may be skipped (with a warning in the log), while the other rules of the same action continue to run.

Related sections

- Overview — diagram elements, running, and management.
- Updating the store from actions — where the rules are enabled.
- Templating — Go template syntax and contexts.

Widgets

Overview

Widgets are cards that visualize data stored in DDP (portal) and information from infrastructure services. Unlike data sources, widgets retrieve information from infrastructure services when they are displayed in the interface.

Widgets can be added to dashboards. Dashboards can be linked to:

- static pages, such as the Catalog, Self-service, Home, and Administration pages;
- entity cards.

Configuration

A widget configuration includes common parameters and fields specific to the widget type.

Widget configurations support Go template syntax for templating during widget processing. For example:

- `{{ .entity.name }}` — substitutes the value of the entity's `name` parameter.
- `{{ .credentials.token }}` — substitutes credentials named `token`.

You can set a scope for each widget:

- `Global` — the widget cannot retrieve entity parameters using Go template.
- `Resource` — the widget can retrieve entity parameters using Go template. Widgets with the `Resource` scope can only be attached to entity pages.

In the widget configuration, you can specify the account whose credentials the widget uses to interact with infrastructure systems and select the credentials type.

If no account is specified, the widget uses the credentials of the current user.

AI

AI chat

The AI chat widget sends requests to a language model through the selected AI provider. You can configure general instructions (the Global prompt) and a set of quick-question buttons (Quick questions).

Configuration

Name	Required	Description
Global prompt	No	General instructions for each request. The widget combines them with the prompt of the selected quick question before sending it
Quick questions	Yes	A set of buttons with labels and prompt text (up to 20). Add at least one question for the widget to work correctly

Configure the following fields for each quick question:

Name	Required	Description
Question name	Yes	A short label displayed in the widget’s bottom panel and in the chat
Prompt	Yes	Instructions sent to the model when the user selects the button. Go templating is supported

When writing a prompt, explicitly specify the names of the Model Context Protocol (MCP) tools that the model must call to prepare the response.

Example prompt for a quick question:

```
1. Call the MCP tool get_external_data for the external service "Deckhouse Code" and
retrieve pipelines for the project with ID {{ .entity.properties.deckhouse_code_id }}.
2. Display a table with the 10 latest pipelines.
```

Using the widget

To use the widget, add at least one AI provider.

The chat does not retain history:

- It displays only one response to the most recent question.
- The response is not retained when the user navigates to another page or refreshes the page.

Before sending a question, the user can customize the prompt by selecting **Send with prompt changes** from the question button menu.

Bitbucket

Bitbucket. Pull Requests

The widget displays Bitbucket Pull Request (PR) data and provides actions for managing PRs.

Configuration

Name	Required	Description	Example
Project key	Yes	The part of the repository URL immediately after `/projects/`	For `https://<BITBUCKET_HOST>/projects/MYTEAM/repos/backend`, specify `MYTEAM`
Repository ID	Yes	The part of the repository URL immediately after `/repos/`	For `https://<BITBUCKET_HOST>/projects/MYTEAM/repos/backend`, specify `backend`

where:

- `<BITBUCKET_HOST>` — the hostname of the Bitbucket server.

Filtering by status

The widget can filter displayed Pull Requests by status. Select one of the following statuses in the widget request settings:

- **Open** — displays only open PRs.
- **Merged** — displays only merged PRs.
- **Declined** — displays only declined PRs.
- **All** — displays PRs in any status.

By default, the widget displays only open PRs.

Additional widget features

When actions are enabled in the settings, the widget provides the following Pull Request actions:

- **Merge** — merges an open Pull Request. This action is available only for open PRs.
- **Close** — declines a Pull Request.
- **View changes** — displays the diff for a Pull Request.
- **Comments** — displays and adds comments to a PR.
- **Create PR** — creates a Pull Request with a source branch, target branch, reviewers, title, and description.

> Pull Request actions require the corresponding permissions in the Bitbucket repository.

Authentication

Authentication is described in External services.

Bitbucket. Tags

The widget displays data about tags in a Bitbucket repository.

Configuration

Name	Required	Description	Example
Project key	Yes	The part of the repository URL immediately after `/projects/`	For `https://<BITBUCKET_HOST>/projects/MYTEAM/repos/backend`, specify `MYTEAM`
Repository	Yes	The part of the repository URL immediately after `/repos/`	For `https://<BITBUCKET_HOST>/projects/MYTEAM/repos/backend`, specify `backend`

where:

- `<BITBUCKET_HOST>` — the hostname of the Bitbucket server.

Displayed data

For each repository tag, the widget displays the tag name and commit details, including the hash, message, author, creation date, and a link to the commit in Bitbucket.

Additional widget features

Creating tags

The widget can create tags in Bitbucket directly from Deckhouse Development Portal (DDP).

Configuration

Name	Required	Description
Name	Yes	The name of the tag to create
Create from	Yes	The branch or existing tag from which to create the new tag. Select it from the list
Description	No	The description of the tag to create

Authentication

Authentication is described in External services.

ClickHouse

ClickHouse. Metrics (range)

The widget plots a line chart based on a read-only SQL query to ClickHouse. Use the `{{from}}` and `{{to}}` placeholders in the query to specify the time range boundaries.

Example of a valid widget query:

```
SELECT
  toStartOfMinute(timestamp) AS time,
  avg(value) AS value,
  service AS series
FROM metrics
WHERE timestamp >= {{from}} AND timestamp < {{to}}
GROUP BY time, series
ORDER BY time
```

Configuration

Name	Required	Description	Default value
Query	Yes	Read-only SQL query. Use `{{from}}` and `{{to}}` to specify the time range	—
Database	No	ClickHouse database name passed in the `X-ClickHouse-Database` header	—
Default range	No	Range used when opening or refreshing the widget if no range is specified in the query parameters	Last hour
Time column	Yes	Column containing timestamps for the chart's X-axis	—
Value column	Yes	Column containing numeric values for the chart's Y-axis	—
Series column	No	Column used as the series name in the chart legend	—
Threshold	No	Threshold displayed as a horizontal line on the chart	—
Minimum value	No	Starting point for the chart's vertical axis	—
Maximum value	No	End point for the chart's vertical axis	—

Authorization

Authorization is described in External services.

ClickHouse. Metrics (single value)

The widget displays a single number based on a read-only SQL query to ClickHouse. You can specify a unit and configure a threshold for the value. The query must return one row; the widget displays the value from the first column.

Example of a valid widget query:

```
SELECT count() AS value
FROM events
WHERE timestamp >= {{from}} AND timestamp < {{to}}
```

Configuration

Name	Required	Description	Default value
Query	Yes	Read-only SQL query. Use `{{from}}` and `{{to}}` to specify the time range	—
Database	No	ClickHouse database name passed in the `X-ClickHouse-Database` header	—
Default range	No	Range used when opening or refreshing the widget if no range is specified in the query parameters	Last hour
Decimal places	No	Precision used to display the returned value	—
Unit	No	Suffix displayed with the returned value	—
Show threshold	No	Displays `<METRIC_VALUE> / <THRESHOLD>`, where `<METRIC_VALUE>` is the current metric value and `<THRESHOLD>` is the configured threshold	`false`
Threshold	No	Threshold value	—
Lower value is better	No	Considers the metric healthy when its value is below the configured threshold	`false`
Warning threshold (%)	No	Boundary between red and orange. A metric value above this percentage of the	60

Name	Required	Description	Default value
		threshold is displayed in orange	
Success threshold (%)	No	Boundary between orange and green. A metric value above this percentage of the threshold is displayed in green	90

Authorization

Authorization is described in External services.

ClickHouse. Table

The widget displays the result of a read-only SQL query to ClickHouse as a sortable, paginated table. Use the `{{from}}` and `{{to}}` placeholders in the query.

Configuration

Name	Required	Description	Default value
Query	Yes	Read-only SQL query. Use `{{from}}` and `{{to}}` to specify the time range	—
Database	No	ClickHouse database name passed in the `X-ClickHouse-Database` header	—
Default range	No	Range used when opening or refreshing the widget if no range is specified in the query parameters	Last hour
Page size	Yes	Number of rows loaded by each query	50

Authorization

Authorization is described in External services.

ClickHouse. Top N

The widget plots a horizontal bar chart based on a read-only SQL query to ClickHouse. The query must return columns containing labels and numeric values. The **Limit** parameter restricts the number of displayed rows.

Example of a valid widget query:

```
SELECT service AS label, count() AS value
FROM events
WHERE timestamp >= {{from}} AND timestamp < {{to}}
GROUP BY label
ORDER BY value DESC
```

Configuration

Name	Required	Description	Default value
Query	Yes	Read-only SQL query. Use `{{from}}` and `{{to}}` to specify the time range	—
Database	No	ClickHouse database name passed in the `X-ClickHouse-Database` header	—
Default range	No	Range used when opening or refreshing the widget if no range is specified in the query parameters	Last hour
Label column	Yes	Column containing the bar labels	—
Value column	Yes	Column containing numeric values that determine bar length	—
Limit	Yes	Maximum number of rows in the chart	10

Authorization

Authorization is described in External services.

CodeScoring

CodeScoring. Dependencies

The widget displays a table of product dependencies based on CodeScoring data. For each dependency, the table includes its name, version, license, number of vulnerabilities, and other information.

Configuration

Name	Required	Description	Default value
URL	Yes	CodeScoring URL	—
Project ID	Yes	Project ID in CodeScoring	—

Authentication

Authentication is described in External services.

CodeScoring. Vulnerabilities

The widget displays a table of product vulnerabilities based on CodeScoring data. For each vulnerability, the table includes its code, severity, exploit availability, and fixed version.

Configuration

Name	Required	Description	Default value
URL	Yes	CodeScoring URL	—
Project ID	Yes	Project ID in CodeScoring	—

Authentication

Authentication is described in External services.

CodeScoring. Secrets

The widget displays a table of secrets detected in a project by CodeScoring. It can start or cancel secret scanning for the selected branch or tag.

Configuration

Name	Required	Description	Default value
URL	Yes	CodeScoring URL	—
Project ID	Yes	Project ID in CodeScoring	—

Authentication

Authentication is described in External services.

DefectDojo

DefectDojo. Product

The widget displays product vulnerabilities from DefectDojo, grouped by engagement and severity.

Configuration

Name	Required	Description	Default value
URL	Yes	DefectDojo URL without the API path (`/api/v2`)	—
Product name	Yes	Product name in DefectDojo	—
Severity levels	Yes	Vulnerability severity levels loaded when the widget opens or when the selected engagement changes	`Critical` , `High` , `Medium` , `Low` , `Info`

Request parameters

In the widget request settings, select the severity levels to load only vulnerabilities with those levels. By default, the widget uses the levels from its configuration. Changing the levels reloads data from DefectDojo.

Additional widget features

Filters

- **Engagement** — selects a product engagement. By default, the most recently created engagement, with the highest ID, is selected. Changing the engagement reloads the data.

- **Filters** — filters vulnerabilities by severity, tags, tests, and components. Tag, test, and component filters apply only to already loaded vulnerabilities and do not send new requests to DefectDojo.

Tabs

- **Overview** — displays vulnerabilities by severity, tags (top 10), tests (top 10), and components (top 10). Charts use the filtered vulnerability set.
- **Details** — displays a table with details for each vulnerability.

> If the selected Engagement and severity levels contain more than 1,000 vulnerabilities, the widget displays the first 1,000 records and a partial-load warning. Tag, test, and component filters apply only to the loaded set.

Authentication

Authentication is described in External services.

DefectDojo. Product vulnerability details

The widget displays a table of product vulnerabilities based on DefectDojo data. For each vulnerability, the table includes its severity, description, and detection date.

Configuration

Name	Required	Description	Default value
URL	Yes	DefectDojo URL without the API path (`/api/v2`)	—
Product name	Yes	Product name in DefectDojo	—

Additional widget features

When viewing the widget, configure the following parameters:

- **Active vulnerabilities** — when enabled, loads product vulnerabilities with the `Active` flag set to `true`. When disabled, loads vulnerabilities with the `Active` flag set to `false`. Enabled by default.
- **Duplicate vulnerabilities** — when enabled, loads product vulnerabilities with the `Duplicate` flag set to `true`. When disabled, loads vulnerabilities with the `Duplicate` flag set to `false`. Disabled by default.

Authentication

Authentication is described in External services.

DefectDojo. Product vulnerabilities summary

The widget displays a chart with the total number of product vulnerabilities from DefectDojo, grouped by severity.

Configuration

Name	Required	Description	Default value
URL	Yes	DefectDojo URL without the API path (`/api/v2`)	—
Product name	Yes	Product name in DefectDojo	—

Additional widget features

When viewing the widget, configure the following parameters:

- **Active vulnerabilities** — when enabled, loads product vulnerabilities with the `Active` flag set to `true`. When disabled, loads vulnerabilities with the `Active` flag set to `false`. Enabled by default.
- **Duplicate vulnerabilities** — when enabled, loads product vulnerabilities with the `Duplicate` flag set to `true`. When disabled, loads vulnerabilities with the `Duplicate` flag set to `false`. Disabled by default.

Authentication

Authentication is described in External services.

Entities

Entity calendar

The widget displays entities of the selected resource in a calendar.

Displayed data

- **Weekly calendar** — a seven-day grid for the current week.
- **Entities by date** — all entities whose selected date field matches a given day.
- **Entity information** — the name and description, if specified, of each entity.
- **Week navigation** — buttons for moving to the previous or next week.

Configuration

Name	Required	Description	Default
Resource	Yes	Resource whose entities are displayed in the calendar	—
Date field	Yes	Field containing the date used to display the entity in the calendar. It can be a system field (`createdAt`, `updatedAt`) or a `Date` parameter	—

Notes

- The widget displays the current week, from Monday through Sunday, by default.
- Use the **Previous week** and **Next week** buttons to navigate between weeks.
- Each day displays its date in `DD.MM` format.
- Entities are displayed as cards that link to the corresponding entity page.
- Entities with an empty or zero date are automatically excluded.

Entity Kanban board

The widget displays entities of the selected resource on a Kanban board.

Displayed data

- **Columns** — configured board columns and a **No status** column for entities without a matching state parameter value.
- **Entity cards** — each entity displays its name, description if specified, check status, owner, and update date.
- **Moving cards** — dragging a card between columns updates the entity's state parameter value. This requires permission to modify entities.

Configuration

Name	Required	Description	Default
Resource	Yes	Resource whose entities are displayed on the board	—
State parameter	Yes	Entity parameter that determines the card column. Supported types: `String`, `Number`, `Boolean`, `Enum`, `List`, `Date`, `Percentage`, and `URL`	—
Columns	Yes	List of board columns. For each column, specify a name (the board heading), value (the state parameter value; for `Enum` and `List` parameters, select from the available options), and color (the tag color in the heading). Drag columns to change their order	—

Notes

Entities without a matching state parameter value are displayed in the **No status** column. Cards cannot be moved without permission to modify entities.

Entity table

The widget displays entities created in Deckhouse Development Portal (DDP) as a table.

Configuration

Name	Required	Description	Default
Resource	Yes	Resource whose entities are displayed in the table	—
Show actions	No	Whether to display entity actions, such as running actions and scenarios or deleting entities	`false`

Entity timeline

The widget displays entities of the selected resource on a timeline.

Displayed data

- **Timeline chart** — a horizontal chart where each entity is represented by a bar spanning from its start date to its end date.
- **Entity information** — hovering over a bar displays the entity name and the period’s start and end dates.
- **Sorting** — entities are sorted from oldest at the top to newest at the bottom.

Configuration

Name	Required	Description	Default
Resource	Yes	Resource whose entities are displayed on the timeline	—
Start date field	Yes	Field containing the period start date. It can be a system field (`createdAt`, `updatedAt`) or a `Date` parameter	—
End date field	Yes	Field containing the period end date. It can be a system field (`createdAt`, `updatedAt`) or a `Date` parameter	—

Notes

The widget automatically scales the timeline to display all entities. Entities with invalid dates, where the start date is later than the end date, are automatically excluded.

Entity status

The widget displays the entity status and status check results.

Displayed data

The widget displays the following information.

Overall status

The **progress bar** visualizes the overall entity status, including the percentage of successful checks. The **successful check count** shows the number of successful checks out of all configured status checks.

Check list

The following information is displayed for each status check:

- **Check name** — the name of the check rule.
- **Status** — the check result:
  - **Passed** — the check completed successfully.

    - **Failed** — the check did not pass, but no execution error occurred.
    - **Error** — an error occurred while running the check.

- **Last check time** — the date and time when the check last ran.
- **Error message** — the error text, displayed if the check ended with an error.

Statistics

The bottom of the widget displays summary check statistics:

- **Passed** — the number of successful checks.
- **Failed** — the number of checks that did not pass without execution errors.
- **Error** — the number of checks that ended with an error.

Blocked actions

The widget automatically identifies and displays actions that are unavailable for the entity's current status.

Display conditions:

- The action must be available for the resource associated with the entity.
- The action must have allowed statuses configured.
- The entity's current status must not be in the action's list of allowed statuses.

The widget displays the action name and description, if specified.

Configuration

The widget requires no additional configuration.

To use the widget, configure status checks for the resource associated with the entity. For details, refer to status check configuration.

Notes

If no status checks are configured for the entity, the widget indicates that no checks are available. If status check data is unavailable, the widget indicates that no data is available.

GitHub

GitHub. Actions

The widget displays GitHub Actions runs in a repository. It also displays jobs and artifacts and provides actions for managing them.

Account and action initiator

Requests to GitHub use the token from the credentials of the DDP (portal) user on whose behalf the action is invoked. If **Select an account for the widget** is enabled in the widget settings, the selected portal user's credentials are used instead of the current user's credentials.

When a workflow is started, canceled, or restarted, or when artifacts and logs are accessed, GitHub identifies the GitHub account that owns the token as the initiator. The login displayed in GitHub may differ from the name in the Deckhouse Development Portal (DDP) profile.

Configuration

Name	Required	Description	Example
Repository owner	Yes	Repository owner, either an organization or a user	For `https://github.com/example/my-repo`, specify `example`
Repository	Yes	Repository name without the `.git` suffix	For `https://github.com/example/my-repo`, specify `my-repo`

Request parameters

Configure the following filters in the widget request settings:

- **Branch** — displays only runs from the specified head branch.
- **Event** — displays only runs triggered by the selected event type.
- **Status** — displays only runs with the selected status or conclusion.
- **Workflow** — displays only runs for the selected workflow file.
- **Triggered by** — displays only runs started by the specified GitHub user.
- **Creation date filter** — displays runs created within the specified start and end dates.

Actions

The widget provides the following actions:

- **Run workflow** — manually starts a workflow with the `workflow_dispatch` trigger. Select the workflow and branch or tag. If the input YAML declares parameters, the input parameters are displayed.
- **Restart workflow**, **Restart failed jobs**, and **Cancel workflow** — manage the selected run.
- **Restart job** — restarts a completed job with the `failure` or `cancelled` conclusion.
- **View run details** — displays job logs, artifacts, the run on GitHub, and the job and step tree.

Workflow and artifact actions require the corresponding permissions in the GitHub repository.

Authentication

Authentication is described in External services. In the external service settings, set **URL** to `https://api.github.com`.

GitHub. Pull Requests

The widget displays Pull Requests (PRs) for a GitHub repository. It also provides actions for viewing changes and creating, merging, and closing PRs.

Account and action initiator

Requests to GitHub use the token from the credentials of the DDP (portal) user on whose behalf the action is invoked. If **Select an account for the widget** is enabled in the widget settings, the selected portal user's credentials are used instead of the current user's credentials.

When a Pull Request is created, merged, or closed, GitHub identifies the GitHub account that owns the token as the PR author and action initiator. The login and name displayed in GitHub may differ from the name in the Deckhouse Development Portal (DDP) profile.

Configuration

Name	Required	Description	Example
Repository owner	Yes	Repository owner, either an organization or a user	For `https://github.com/example/my-repo`, specify `example`
Repository	Yes	Repository name without the `.git` suffix	For `https://github.com/example/my-repo`, specify `my-repo`

Status

Filter PRs by status in the widget request settings:

- **Open** — displays only open PRs that are not drafts.
- **Draft** — displays only drafts.
- **Closed** — displays only closed PRs.
- **All** — displays PRs in any status.

By default, the widget displays open PRs. The table displays the number, title, description, status, labels, author, creation date, and update date. An action menu is available for each PR.

Actions

- **Changes** — displays the list of changed files and the diff for each file.
- **Merge** — merges an open PR. This action is available only for open PRs that are not drafts.
- **Close** — closes a PR without merging it.
- **Create PR** — creates a Pull Request. In the dialog, specify the title, source branch, target branch, and description.

Pull Request actions require the corresponding permissions in the GitHub repository.

Authentication

Authentication is described in External services. In the external service settings or widget configuration, set **URL** to `https://api.github.com`.

GitHub. Tags

The widget displays GitHub repository tags with commit information, including the author, date, and description. It can also create tags.

Account and action initiator

Requests to GitHub use the token from the credentials of the DDP (portal) user on whose behalf the action is invoked. If **Select an account for the widget** is enabled in the widget settings, the selected portal user's credentials are used instead of the current user's credentials.

For an annotated tag, where **Description** is provided, the annotation author fields (`tagger`) in the Git tag metadata are populated with the name and email address of the portal user who performed the action, as defined in the Deckhouse Development Portal (DDP) profile. If no name is specified, the email address may be used.

For a lightweight tag without a description, no separate Git tag author is set. The tag is created as a reference to a commit.

The API creates the tag using the GitHub account associated with the token. However, the `tagger` data comes from the DDP profile and may not match the GitHub login.

Configuration

Name	Required	Description	Example
Repository owner	Yes	Repository owner, either an organization or a user	For `https://github.com/example/my-repo`, specify `example`
Repository	Yes	Repository name without the `.git` suffix	For `https://github.com/example/my-repo`, specify `my-repo`

Displayed data

The table displays the tag, description, commit author, commit link, and commit creation date. The **View** action displays the commit description for each tag.

Additional widget features

Creating a tag

The widget can create tags in GitHub. Specify the following fields in the **Create tag** dialog:

Name	Required	Description	Default value
Tag name	Yes	A unique tag name, such as `v1.0.0` or `release-2024-01`	—
Create from	Yes	The branch or existing tag from which to create the new tag	—
Description	No	Tag annotation, such as a release description. If specified, the widget creates an annotated tag	—

Creating tags requires write permissions in the GitHub repository.

Authentication

Authentication is described in External services. In the external service settings, set **URL** to `https://api.github.com`.

GitLab

GitLab. Members

The widget displays data about GitLab project members. Learn more about project members.

Configuration

Name	Required	Description	Default value
Project ID	Yes	ID of the project from which the widget retrieves data. Example: `12345`	—

Authentication

Authentication configuration is described in the External services section.

GitLab. Merge requests

The widget displays GitLab merge requests (MRs) and provides actions for managing them.

Configuration

Name	Required	Description	Default value
URL	Yes	GitLab API URL used to retrieve data from GitLab	—
Project ID	Yes	ID of the project from which the widget retrieves data. Example: `12345`	—

Status filtering

The widget can filter merge requests by status. In the widget request settings, select one of the following statuses:

- **Open** — displays only open MRs.
- **Closed** — displays only closed MRs.
- **Merged** — displays only merged MRs.
- **Blocked** — displays only blocked MRs.

By default, the widget displays only open MRs.

Additional widget features

When actions are enabled in the settings, the widget provides the following merge request actions:

- **Merge** — merges an open merge request. This action is available only for open MRs.
- **Close** — closes a merge request.
- **Mark as draft/ready** — changes the draft status of a merge request.
- **View changes** — displays the diff for a merge request.

Actions on MRs require the corresponding access permissions in the GitLab repository.

Authentication

Authentication configuration is described in the External services section.

GitLab. Pipeline editor

The widget lets you edit the GitLab CI/CD pipeline configuration in `.gitlab-ci.yml` and create merge requests with the changes.

Configuration

Name	Required	Description	Default value
URL	Yes	GitLab API URL used to retrieve data from GitLab	—
Project ID	Yes	ID of the project from which the widget retrieves data. Example: `12345`	—

Displayed data

The widget displays a Monaco code editor for editing `.gitlab-ci.yml` and a diff view of the original and edited configuration.

Additional widget features

Creating a merge request

The widget lets you create a merge request containing pipeline configuration changes.

Merge request parameters

Name	Required	Description	Default value
MR title	Yes	Short title describing the purpose of the merge request	—
MR description	No	Detailed description of the merge request and changes	—
New branch name	Yes	Name of the new branch that will contain the changes	—
Target branch	Yes	Branch to which the merge request will be submitted	`main`
Commit message	Yes	Description of changes to the pipeline configuration	—

Limitations

- The widget supports only the `.gitlab-ci.yml` file in the project root.
- Creating a merge request requires write access to the repository.
- The maximum configuration file size is limited by the GitLab API.

Authentication

Authentication configuration is described in the External services section.

GitLab. Pipeline statistics

The widget displays GitLab pipeline statistics, including overall statistics and breakdowns by status, source, member, and branch.

Configuration

Name	Required	Description	Default value
URL	Yes	GitLab API URL used to retrieve data from GitLab	—
Project ID	Yes	ID of the project from which the widget retrieves data. Example: `12345`	—

Displayed data

The widget displays the following statistics:

Key metrics

- **Total pipelines** — total number of pipelines during the selected period.
- **Success rate** — percentage of successful pipelines.
- **Failure rate** — percentage of failed pipelines.
- **Average duration** — average pipeline execution time.

Breakdown by status

- Successful pipelines.
- Failed pipelines.
- Canceled pipelines.
- Skipped pipelines.
- Manual pipelines.

Breakdown by source

- Push events (commits).
- Merge requests.
- Scheduled runs.
- Web interface.

Top members

- Members who started the most pipelines.
- Member avatars, if available.
- Number of pipelines for each member.

Branch activity

The widget shows the branches with the most pipelines and the number of pipelines for each branch.

Request parameters

Name	Required	Description	Default value
Start date	Yes	Start date of the pipeline analysis period in ISO 8601 format. Example: `2024-01-01T00:00:00Z`	—
End date	Yes	End date of the pipeline analysis period in ISO 8601 format. Example: `2024-01-31T23:59:59Z`	—
Branch	No	Filters by a specific branch. If omitted, the widget analyzes all branches	—

Limitations

- To optimize performance, the widget analyzes no more than 100 pipelines per request.
- Statistics include only pipelines with valid data, including a status and execution time.
- Data is updated each time the widget refreshes.

Authentication

Authentication configuration is described in the External services section.

GitLab. Pipelines

The widget displays data about GitLab pipelines.

Configuration

Name	Required	Description	Default value
URL	Yes	GitLab API URL used to retrieve data from GitLab	—
Project ID	Yes	ID of the project from which the widget retrieves data. Example: `12345`	—

Additional widget features

Starting pipelines

The widget lets you start GitLab pipelines directly from Deckhouse Development Portal (DDP).

Configuration

Name	Required	Description	Default value
Ref	Yes	Target branch or tag on which to start the pipeline	—
Variables	No	Key-value variables to pass to the pipeline being started	—

Authentication

Authentication configuration is described in the External services section.

GitLab. Releases

The widget displays GitLab project releases, highlights the latest release, and shows related information: the tag, commit link, author, publication date, and a description with Markdown support.

Configuration

Name	Required	Description	Default value
Project ID	Yes	ID of the project from which the widget retrieves data. Example: `12345`	—

Additional widget features

Creating a release

The widget lets you create a release in GitLab directly from Deckhouse Development Portal (DDP):

Name	Required	Description	Default value
Release name	Yes	Release name displayed in the list	—
Tag	Yes	Existing tag on which to base the release, selected from the project's tags	—
Description	No	Release description in Markdown format	—

The created release appears in the list automatically, and the latest release is highlighted.

Authentication

Authentication configuration is described in the External services section.

GitLab. Tags

The widget displays data about GitLab project tags.

Configuration

Name	Required	Description	Default value
URL	Yes	GitLab API URL used to retrieve data from GitLab	—
Project ID	Yes	ID of the project from which the widget retrieves data. Example: `12345`	—

Additional widget features

Creating tags

The widget lets you create GitLab tags directly from Deckhouse Development Portal (DDP).

Configuration

Name	Required	Description	Default value
Name	Yes	Name of the tag to create	—
Create from	Yes	Branch or existing tag from which to create the tag	—
Description	No	Description of the tag to create	—

Authentication

Authentication configuration is described in the External services section.

Kafka

Kafka. ACLs

The widget displays a list of access control lists (ACLs) for a Kafka cluster.

For each ACL, the widget displays:

- Principal.
- Resource type.
- Pattern.

- Pattern type.
- Host.
- Operation.
- Permission type.

Configuration

Name	Required	Description	Default value
URL	Yes	Kafka cluster URL	—
Authentication protocol	Yes	Protocol used to connect to Kafka. Authentication protocol reference	—
SASL mechanism	No	Authentication mechanism used by SASL. Required when using the `SASL_PLAINTEXT` or `SASL_SSL` protocol. SASL reference	—
Kafka user	Yes	Username of the account used to interact with Kafka	—
Password	Yes	Password of the account used to interact with Kafka	—
Resource types	No	Filter by resource type	—
Pattern types	No	Filter by pattern type	—
Operations	No	Filter by operation	—
Permission types	No	Filter by permission type	—
Principals	No	Filter by principal. Supports templates and regular expressions	—
Hosts	No	Filter by host. Supports templates and regular expressions	—

Additional widget capabilities

When actions are enabled in the settings, the widget allows users to create and delete ACL rules.

Authentication

The widget requires a user account. The system supports the following authentication methods:

- `PLAINTEXT`.
- `SCRAM-SHA-256`.
- `SCRAM-SHA-512`.

Kafka. Topics

The widget displays Kafka topic data.

The following information and actions are available for each topic:

- General topic information, including key parameters, configuration, and status.
- Partition information, including leaders, offsets, and replica counts.
- Consumer information, including active consumers, their groups, current offsets, and lag.
- Message content.
- Message search by timestamp and offset.
- Topic configuration as a key-value table.

Configuration

Name	Required	Description	Default value
URL	Yes	Kafka cluster URL	—
Authentication protocol	Yes	Protocol used to connect to Kafka. Authentication protocol reference	—
SASL mechanism	No	Authentication mechanism used by SASL. Required when using the `SASL_PLAINTEXT` or `SASL_SSL` protocol. SASL reference	—
Kafka user	Yes	Username of the account used to interact with Kafka	—
Password	Yes	Password of the account used to interact with Kafka	—
Kafka topics	No	Topic name or regular expression used to filter topics displayed in the widget. If empty, all topics available to the user are displayed	—

Additional widget capabilities

When actions are enabled in the settings, the widget allows users to:

- Create topics.
- Delete topics.
- Send a message to a topic.
- Remove all messages from a topic.

Authentication

The widget requires a user account. The system supports the following authentication methods:

- `PLAINTEXT`.
- `SCRAM-SHA-256`.
- `SCRAM-SHA-512`.

> The connected account's permissions determine which information is available in the widget.

Kaiten

Kaiten. Space cards

The widget displays the task structure of a Kaiten space as a multi-level **Board** → **Cards** table. Use it to view tasks at every level of the work hierarchy and review key card details, including status, urgency, blocking state, and assignees.

Configuration

Name	Required	Description	Default
Space ID	Yes	Kaiten space identifier	—

Query parameters

Name	Required	Description	Default
My tasks	No	Filters by the current user	`false`
Created after	Yes	Start date of the query period	1 month ago
Created before	Yes	End date of the query period	Now

Displayed data

Each card contains:

- Card name.
- Column (board status).
- Status (queued, in progress, or completed).
- Lane.
- Owner (avatar, name, and email).
- Participants.

- Due date and urgency.
- Blocking state.

Authorization

Authorization is described in External services.

Kaiten. Space statistics

The widget provides aggregated metrics and statistics for cards in a Kaiten space over a selected period. Use it to analyze team performance and identify bottlenecks in business processes.

Configuration

Name	Required	Description	Default
Space ID	Yes	Kaiten space identifier	—

Query parameters

Name	Required	Description	Default
Created after	Yes	Start date of the analysis period	1 month ago
Created before	Yes	End date of the analysis period	Now

Displayed data

The widget contains four tabs.

General metrics

Primary metrics:

- **Queued** — tasks waiting to be processed.
- **Completed** — completed tasks.
- **In progress** — active tasks.

Additional metrics:

- **Blocked** — number of blocked tasks.
- **Blocking** — number of tasks that block other tasks.
- **Archived** — number of archived tasks.
- **Urgent** — number of urgent tasks.
- **Average completion time** — average time to complete a task, in minutes.

Checklist statistics:

- **Total with checklists** — total number of tasks that have checklists.
- **Checklist completed** — tasks with fully completed checklists.
- **Checklist incomplete** — tasks with incomplete checklists.

By user

- List of users and their assigned task counts.
- Progress bar visualization.
- Number of tasks assigned to each user.

Forgotten tasks

Cards that have not been updated since they were created.

Recently updated

The ten most recently updated cards.

Authorization

Authorization is described in External services.

Kubernetes

Kubernetes. Deployments

The Kubernetes deployments widget displays key information about all Deployments in a Kubernetes cluster. You can filter Deployments by namespace, label selector, or both.

The following actions are available for each Deployment:

- View the Deployment specification and status.
- Scale the number of Deployment replicas. After selecting the required number of replicas, apply the change by clicking the floppy-disk **Save** button.
- View information about pods managed by the Deployment and their containers, including logs for each container.
- View and edit container resources. The widget displays all configured container resources, including CPU, memory, `ephemeral-storage`, and other resource types. You can edit only CPU and memory in the `requests` and `limits` sections. Changes are applied at the Deployment level and propagated to all pods managed by that Deployment. Clearing a CPU or memory value removes the corresponding resource from the container configuration. Other resources, such as `ephemeral-storage`, are displayed but cannot be edited in the widget.

Configuration

Name	Required	Description	Default value
Kubernetes API	Yes	Kubernetes API server URL used to retrieve data from Kubernetes	—
Namespace	No	Kubernetes namespace from which Deployments are loaded. If no namespace is specified, the widget attempts to load all Deployments in the cluster. Example: `default`	—
Label selector	No	Comma-separated selectors used to filter Deployments. Example: `app.kubernetes.io/name=example`	—

Authorization

Authorization is described in External services.

Kubernetes. Ingresses

The widget displays information about Ingress resources in a Kubernetes cluster.

The following information is available for each Ingress:

- Ingress specification as YAML configuration.
- Ingress rules.
- TLS settings.

Configuration

Name	Required	Description	Default value
URL	Yes	Kubernetes API server URL used to retrieve data from Kubernetes	—
Namespace	No	Kubernetes namespace from which Ingress resources are loaded. If no namespace is specified, the widget attempts to load all Ingress resources in the cluster. Example: `default`	—
Label selector	No	Comma-separated selectors used to filter Ingress resources. Example: `app.kubernetes.io/name=example`	—

Authorization

Authorization is described in External services.

Kubernetes. Pods

The widget displays information about pods in a Kubernetes cluster.

The following information is available for each pod:

- Pod specification as YAML configuration.
- Container logs.
- Pod state information, including status and restart count.

Configuration

Name	Required	Description	Default value
URL	Yes	Kubernetes API server URL used to retrieve data from Kubernetes	—
Namespace	No	Namespace from which the widget loads data. Example: `default`	—
Label selector	No	Comma-separated selectors used to filter pods. Example: `app.kubernetes.io/name=example`	—

Authorization

Authorization is described in External services.

Kubernetes. Resource quotas

The widget displays resource quota data from a Kubernetes cluster.

For each quota, the widget visualizes the resources in use.

Configuration

Name	Required	Description	Default value
URL	Yes	Kubernetes API server URL used to retrieve data from Kubernetes	—
Namespace	Yes	Namespace from which the widget loads data. Example: `default`	—

Authorization

Authorization is described in External services.

Prometheus

Prometheus. Metrics (range)

The widget plots a chart from the result of a Prometheus `query_range` query. The PromQL query must return a `Matrix` type containing one or more time series.

Example query with placeholders:

```
sum by (status) (rate(http_requests_total[{{rateInterval}}]))
```

Query placeholders

The widget substitutes values calculated from the active time range:

Placeholder	Description
`{{range}}`	Duration of the selected time range
`{{rateInterval}}`	Range window for the `rate()` and `increase()` functions
`{{interval}}`	Interval between chart points, calculated automatically

The query step is selected based on the time range duration. The chart contains no more than 1100 points, and the minimum step is 30 seconds. The **Resolution step** configuration field is no longer used.

Configuration

Name	Required	Description	Default value
URL	Yes	Prometheus URL	—
Query	Yes	PromQL query for the range chart	—
Label	Yes	Prometheus label name used as the series name in the chart legend	—
Default range	No	Range used when loading or refreshing the widget unless the user selects another range in the widget panel	Last hour
Threshold	No	Horizontal dashed line on the chart	—
Minimum value	No	Lower Y-axis boundary. Leave empty to scale automatically	—
Maximum value	No	Upper Y-axis boundary. Leave empty to scale automatically	—
`InsecureSkipVerify`	No	Disables verification of the Prometheus TLS/SSL certificate	`false`

Authorization

Authorization is described in External services.

Prometheus. Metrics (single value)

The widget displays a single number from a PromQL query to Prometheus. The query must return a `Scalar` or a `Vector` containing one value.

Example query without placeholders:

```
sum(machine_cpu_cores)
```

Example with a placeholder for the time range duration selected in the widget panel:

```
sum(increase(http_requests_total[{{range}}]))
```

Query placeholders

Use placeholders when the expression requires the duration of the time range selected in the widget panel. The query runs at the end of this time range.

Placeholder	Description
`{{range}}`	Duration of the selected time range
`{{rateInterval}}`	Range window for the `rate()` and `increase()` functions
`{{interval}}`	Query step calculated from the time range

Configuration

Name	Required	Description	Default value
URL	Yes	Prometheus URL	—
Query	Yes	PromQL query that returns one value	—
Default range	No	Range used when loading or refreshing the widget unless the user selects another range in the widget panel	Last hour
Decimal places	No	Precision used to display the returned value	—
Unit	No	Suffix displayed with the returned value	—
Show threshold	No	Displays `<METRIC_VALUE> / <THRESHOLD>`, where `<METRIC_VALUE>` is the current metric value and `<THRESHOLD>` is the configured threshold	`false`
Threshold	No	Threshold value	—
Lower value is better	No	Considers the metric healthy when its value is below the configured threshold	`false`
Warning threshold (%)	No	Boundary between red and orange. A metric value above this percentage of the threshold is displayed in orange	60
Success threshold (%)	No	Boundary between orange and green. A	90

Name	Required	Description	Default value
		metric value above this percentage of the threshold is displayed in green	
`InsecureSkipVerify`	No	Disables verification of the Prometheus TLS/SSL certificate	`false`

Authorization

Authorization is described in External services.

Other

API

The widget displays an API specification from a file in a GitLab repository or from a URL in OpenAPI (Swagger) or Protobuf format. For an OpenAPI specification in a YAML or JSON file, the widget displays the Swagger UI. In all other cases, it displays the specification as text.

General configuration

Name	Required	Description	Possible values	Default
Specification type	Yes	Specification type	OpenAPI (Swagger), Protocol Buffers	—
Source type	Yes	Source from which the specification file is loaded	URL, GitLab	—

Source type configuration: URL

Name	Required	Description	Default
URL	Yes	URL of the specification file	—
Headers	No	Headers used to access the specification file	—

Source type configuration: GitLab

Name	Required	Description	Default
GitLab URL	Yes	GitLab URL	—
Project ID	Yes	ID of the project containing the specification file	—
Branch	Yes	Branch containing the specification file	—
File path	Yes	Path to the specification file relative to the repository root	—

Authorization

Authorization is configured in External services.

Helm releases

The widget displays data about Helm releases in Kubernetes and lets you roll back to previous versions.

The widget displays:

- **Helm release list** — Information about current releases created with Helm in the specified Kubernetes namespace.
- **Release manifests** — Manifests associated with Helm releases in the specified Kubernetes namespace, including YAML files that define resource configuration and state.
- **Values** — Variables used to deploy Helm releases.

Configuration

Name	Required	Description	Default
Namespace	No	Namespace from which the widget loads data. Example: `default`	—
Release	No	Name of the release from which the widget loads data. Example: `my-release`	—

Authorization

Authorization is configured in External services.

SonarQube

The widget displays SonarQube metrics.

Configuration

Name	Required	Description	Default
URL	Yes	SonarQube URL, for example, `https://sonarqube.example.com`	—
Project key	Yes	Project identifier in SonarQube	—
Branch	No	Project branch from which metrics are retrieved	According to the SonarQube project settings
Metrics	Yes	Project metrics displayed in the widget. Specify each metric key in the configuration	—

See the list of available metrics for the current SonarQube version.

Additional widget features

The widget can display data for the default branch or any other branch.

Authorization

Authorization is configured in External services.

OpenSearch

The OpenSearch index widget displays data from a specific index or index pattern in the portal. By default, data is sorted from newest to oldest. Full-text search is available to filter the displayed data. Each record (table row) can be displayed as key-value pairs or as JSON. When an index pattern is specified, the widget provides a link to the Discover page in OpenSearch Dashboards.

Configuration

Name	Required	Description	Default
API URL	Yes	OpenSearch API URL used to retrieve data	—
Dashboards URL	Yes	OpenSearch Dashboards URL used to generate a link for viewing data directly in OpenSearch	—
Index pattern	Yes	Index pattern from which the widget loads data. May contain `*`. Examples: `security-auditlog`, `security-auditlog-*`	—
Timestamp field	No	Name of the timestamp field. Its value is displayed in a separate column in the data table	`@timestamp`

Authorization

Authorization is configured in External services.

Svacer

The widget displays static code analysis results for a project branch in Svacer: marker review progress, findings by severity, branch finding trends, snapshot comparisons, and a paginated marker table.

Configuration

Name	Required	Description	Default
Project name	Yes	Project name in Svacer	—
Branch name	Yes	Branch name in Svacer, used as the default branch when the widget opens	—
Cache the full marker list	No	Caches the selected Svacer snapshot's full marker list in the DDP backend to speed up pagination on the **Findings** tab	Enabled
Cache lifetime (seconds)	No	Number of seconds to keep the response in DDP memory when caching is enabled. Valid range: 30–86400	180
Snapshot marker threshold	No	Number of markers above which a snapshot is considered large	10,000
Large snapshot strategy	No	Behavior when the marker threshold is exceeded	Hybrid

Large snapshot strategies:

- **Hybrid** — The unfiltered **Findings** tab is unavailable. The overview is generated using lightweight Svacer requests.
- **Unlimited** — The full marker list is always loaded. For a large snapshot, the widget only displays a warning.

Query parameters

The following parameters are available when viewing the widget:

- **Branch** — Svacer branch from which data is loaded. The list is generated for the project in the widget configuration. The configured branch is selected by default.

- **Snapshot** — Branch snapshot from which data is loaded. **Latest snapshot** refers to the current branch snapshot when the widget is refreshed.
- **Filters**:
  - **Review status** — Marker review status: confirmed, false positive, unclear, no decision, or will not fix.
  - **Severity** — Finding severity.
  - **Checker** — Checker name, for example, `gosec.G402`.

Tabs

- **Overview** — Review progress, number of unreviewed markers, branch findings, findings by severity, finding trends, reviews by status, and comparison with the previous snapshot: new, fixed, matched, and unchanged.
- **Findings** — Paginated marker table with **Severity**, **Confidence**, **Location**, **Checker**, **Review**, **Message**, and **Tags** columns. Each marker links to Svacer.

The widget footer displays summary metrics for the selected snapshot: total markers, unreviewed markers, and filtered findings by severity.

> With the **Hybrid** strategy, if the snapshot marker count exceeds the threshold, the **Findings** tab is unavailable unless you filter by review status, severity, or checker.

Authorization

Authorization is configured in External services.

Vault secrets

The widget lets you browse secrets in HashiCorp Vault or Deckhouse Stronghold. KV v2 secrets are supported.

> The widget does not send secret values to users. Only secret metadata, such as version and creation time, and the key structure without values are sent to the client.

For each secret, you can:

- Browse the hierarchical structure of secrets and directories.
- View secret metadata: version, creation time, deletion time, and destruction status.
- View secret keys in a key-value table. Placeholders are displayed instead of values.
- Navigate nested secrets and directories.

Configuration

Name	Required	Description	Default
Path	Yes	Path to a secret or directory in Vault. The path must explicitly include `/data/`. Examples: `services/data/`, `services/data/example`	—
UI prefix	No	UI URL prefix. Use `vault` for HashiCorp Vault or `stronghold` for Deckhouse Stronghold	—

Path behavior

Paths to KV v2 secrets must explicitly include `/data/`. Valid examples:

- `services/data/` — Browse all secrets in the `services` directory.
- `services/data/example` — View the `example` secret.
- `services/data/nested/secret` — View a nested secret.

Displayed data

The widget displays the following information.

Secret structure

- **Directories** — Displayed with a trailing slash, for example, `nested/`, and always identified as directories even if they contain keys.
- **Secrets** — Displayed without a trailing slash and contain keys.

Secret metadata

The following metadata is displayed when available:

- **Version** — Secret version in KV v2.
- **Creation time** — Date and time when the secret was created.
- **Deletion time** — Date and time when the secret version was deleted.
- **Destruction status** — Indicates that the secret was destroyed.

Secret keys

Secret keys are displayed in a key-value table:

- **Key** — Full path to the key in the secret structure, for example, `database.host`.
- **Value** — Always masked as `********` and cannot be revealed.

Authorization

Authorization is configured in External services.

Docker

The widget displays available images in a Docker registry. It shows all available tags and the `docker pull` command. Search is supported.

Configuration

Name	Required	Description	Default
URL	Yes	Docker Registry URL used to retrieve available image data	—
Name	No	Name of the repository from which the widget loads data. Example: `repo`. If omitted, all available images are loaded	—

Authorization

Authorization is configured in External services.

Jenkins

The widget displays Jenkins pipeline data and lets you manage builds.

Configuration

Name	Required	Description	Default
URL	Yes	Jenkins URL used to retrieve data	—
Name	Yes	Jenkins pipeline name. Nested paths are supported: `folder1/folder2/jobName`	—

Displayed data

The widget automatically detects the pipeline type and displays the appropriate view.

Regular pipelines

For regular pipelines, the widget displays:

- **Build list** — A table of all pipeline builds with their number, status, duration, execution time, and user.
- **Latest build** — Information about the latest completed build.
- **Latest successful build** — Information about the latest successful build.
- **Latest failed build** — Information about the latest failed build.

Multibranch pipelines

For multibranch pipelines, the widget displays:

- **Branch list** — A table of all branches with their status, build count, and latest build.
- All information described for regular pipelines, grouped by branch.

Additional widget features

The widget supports the following actions.

Regular pipelines

- **Start build** — Starts a new build. If the build has parameters, the widget opens a dialog for entering:
  - String parameters.
  - Passwords.
  - List selections.
  - Boolean values.
- **Cancel build** — Cancels a running build.
- **Rebuild** — Runs the latest build again.

- **View logs** — Displays build execution logs.

Multibranch pipelines

- **Start branch build** — Starts a new build for a specific branch. If the build has parameters, the widget opens a dialog for entering them.
- **Get branch builds** — Loads the build list for a specific branch.
- **Scan multibranch** — Starts a multibranch pipeline scan to discover new branches.
- **View logs** — Displays build execution logs.

Authorization

Authorization is configured in External services.

Nexus

The widget displays a list of artifacts in a Nexus repository.

Configuration

Name	Required	Description	Default
URL	Yes	Nexus API URL used to retrieve data	—
Repository	Yes	Name of the repository whose data is displayed in the widget. Example: `my-repo`	—
Name	No	Name of the artifact whose data is displayed in the widget	—

Authorization

Authorization is configured in External services.

S3

The widget lets you browse S3-compatible object storage, including Amazon S3, Yandex Object Storage, and MinIO.

For each object, you can:

- View objects in a bucket with their size, modification date, and storage class.
- Search for objects by prefix (path).

- Download files from the bucket.
- View object metadata, including size, content type, custom metadata, and cache settings.
- Navigate bucket folders.

Using credential templates

You can use credential templating:

- `{{ .credentials.accessKeyId }}` — Inserts the Access Key ID from the credentials.
- `{{ .credentials.secretAccessKey }}` — Inserts the Secret Access Key from the credentials.

> The connected account's permissions determine what information is available in the widget.

Configuration

Name	Required	Description	Default
Bucket name	Yes	Name of the S3 bucket to browse	—
Endpoint	Yes	S3-compatible storage endpoint URL, for example, `https://storage.yandexcloud.net`	—
Region	Yes	Region containing the bucket	—
Access Key ID	Yes	Access key identifier used for authentication	—
Secret Access Key	Yes	Secret access key used for authentication	—
Prefix	No	Prefix (path) used to filter objects on initial load	—
Maximum objects	No	Maximum number of objects displayed per request	100

Additional widget features

Object search

The widget searches for objects by prefix (path). The object list updates to match the specified prefix.

> Search matches only characters at the beginning of an object name. Searching by characters in the middle of a name is not supported.

If the widget configuration specifies an initial prefix, searches are restricted to that prefix. Files with other prefixes cannot be loaded or displayed.

Downloading files

The widget lets you download files from a bucket directly in the browser. A download button is available for each file.

Object details

Select the document icon for an object to view:

- Basic properties: key, size, modification date, storage class, content type, and ETag.
- Content information: encoding, language, and disposition.
- Cache settings: Cache-Control and expiration.
- Security: server-side encryption.
- Custom metadata.

Loading more objects

For buckets containing many objects, use **Load more** to load objects incrementally without reducing performance.

Authentication

The widget requires credentials with access to the S3 bucket. It supports:

- Access Key ID and Secret Access Key.
- Different S3-compatible providers through endpoint configuration.

> Unlike other widgets, the S3 widget cannot obtain credentials from external services. Specify all authentication parameters directly in the widget configuration.

Jira

The widget displays Jira issues based on a JQL query.

Configuration

Name	Required	Description	Default
URL	Yes	Jira URL used to retrieve data	—
JQL	Yes	JQL query used to filter issues. Example: `project = PROJ AND status = Open`	—

Query parameters

Name	Required	Description	Default
JQL	No	JQL query used to filter issues. If omitted, the configured JQL is used	From configuration
Maximum results	No	Maximum number of issues to display, from 1 to 1000	50

Additional widget features

- **View description** — Opens a dialog with the full issue description.
- **Open in Jira** — Opens the issue in a new tab when you select its key.
- **Dynamic filtering** — Lets you change the JQL query and maximum number of results directly in the widget without changing its configuration.

Authorization

Authorization is configured in External services.

Iframe

> ⚠ The Iframe widget works only when `allowIframe: true` is enabled in the security headers configuration (`security.headers.csp.allowIframe`). This option is disabled by default, so the widget does not display content until the configuration is changed.

The widget displays data from an external source.

Configuration

Name	Required	Description	Default
URL	Yes	External source URL used to display data in the widget	—

Graph

The widget displays information about DDP objects using one of the following chart types:

- Bar chart.
- Doughnut chart.
- Pie chart.
- Polar area chart.
- Radar chart.

Configuration

Name	Required	Description	Default
Chart type	Yes	Chart visualization type	—
Table name	Yes	Database table containing the records to visualize	—
Field name	Yes	Field used to aggregate records	—
Filters	No	Fields and values used to filter the retrieved records	—
Aggregation type	Yes	Method used to group the retrieved records	—
Aggregation parameters	No	Time range and grouping step used when aggregating records by date	—

When configuring the widget, account for differences between database field names and object specification field names. When structures are stored in the database, camelCase names from object specifications are converted to snake_case. For example:

- Specify the `createdAt` field as `created_at` in the widget configuration.
- Specify the `resourceUuid` field as `resource_uuid` in the widget configuration.

Nested values are supported. Separate nesting levels with a period. For example, configure the widget as follows to aggregate entities by status:

Table name	Field name
`entities`	`health.status`

Aggregation types

Date

Chart data is sorted and grouped by the selected time intervals.

You can configure the following aggregation parameters:

- **Step unit** — For example, seconds, minutes, hours, or days.
- **Units per step** — For example, 5 minutes, 2 hours, or 1 day.

These parameters control the time-based granularity of the displayed data.

Value

Chart data is sorted by value. For each unique value in the source data:

- The number of occurrences is counted.
- The chart displays a value-count pair.

This aggregation shows the distribution and frequency of values.

Interval ranges

The **Interval ranges** aggregation divides values into configured numeric ranges. Use it to build histograms and analyze data distributions.

Configure intervals in one of two modes:

1. **Automatic division by interval count**.

   Specify the number of intervals into which the available data is divided. Intervals are calculated automatically and distributed evenly from the minimum to the maximum value.

2. **Manual interval boundaries**.

   Specify an array of numeric interval boundaries. For example: `0, 10, 20, 50`.

The numbers are sorted in ascending order and form these intervals: `[0, 10)`, `[10, 20)`, `[20, 50]`.

Specify at least one of these aggregation parameters:

- `Count` — Number of intervals.
- `Boundaries` — Interval boundaries.

Examples:

- `Count = 5` — Creates five equal intervals based on the data.
- `Boundaries = 100, 0, 50` — Sorts the boundaries to `[0, 50, 100]` and creates the intervals `[0, 50)` and `[50, 100]`.

Event statistics

The widget displays statistics about events involving DDP entities. It contains three tabs:

1. **Event statistics** — A chart showing the number of events by type over the selected time range, with configurable time grouping.
2. **Top entities** — A table of entities that generated the most events.
3. **Redis events** — A table of Redis event streams. For each stream, it shows:
    - The stream name, which you can select to view all events.
    - The resource associated with the stream.
    - The number of events in the stream.
    - Information about the latest event: entity, resource, event type, and time.

Query parameters

Name	Required	Description	Default
Date from	Yes	Start date for selecting events	3 days ago
Date to	Yes	End date for selecting events	Current date
Interval	No	Chart grouping interval: seconds, minutes, hours, days, weeks, months, or years	Hour
Interval step	No	Number of interval units used for grouping	1
Top entities	No	Number of entities with the most events to display in the table	10

Event types

The widget supports the following event types:

- `ENTITY_CREATED` — Entity created.
- `ENTITY_UPDATED` — Entity updated.
- `ENTITY_DELETED` — Entity deleted.

Behavior

- The chart shows events over the selected time range with configurable time grouping. The default grouping is by hour.
- The table displays all events for each entity without date filtering.
- Deleted entities are shown using the names extracted from their event specifications.
- The **Redis events** tab lets you monitor events stored in Redis Streams:
  - Each stream shows its event count and latest event.
  - Selecting a stream name opens a dialog containing all events from that stream.
  - Streams are automatically associated with resources by the UUID in the stream name.
  - The stream view shows the latest 1,000 events, newest first. Older events are not displayed.
- Each table row contains information about the latest event for the entity.
- A detailed change history is available for each entity.

- Events for deleted resources are not displayed because they are removed from the database when the resource is deleted.

Task queue

The widget monitors the task queue and the workers that process tasks in the background. It displays queue statistics, worker (consumer) information, and details of every task in the queue.

Displayed data

The widget has three main sections.

Queue statistics

The top of the widget displays four key metrics:

- **Queue size** — Total number of tasks in the queue.
- **Pending tasks** — Number of tasks waiting to be processed.
- **Active workers** — Number of active workers (consumers) processing tasks.
- **Queued tasks** — Total number of new and in-progress tasks.

Workers table

The table provides information about each active worker:

- **Consumer name** — Worker (consumer) identifier.
- **Pending tasks** — Number of tasks assigned to the worker and waiting to be processed.
- **Idle time** — Time since the worker's last activity.

> The table displays only active workers. Workers that are not processing tasks and have been inactive for more than five minutes are automatically hidden.

Tasks table

The table provides details about every task in the queue:

- **Task UUID** — Unique task identifier.
- **Type** — Task type, for example, `health_check`.
- **Resource UUID** — Identifier of the resource or entity associated with the task.
- **Consumer** — Name of the consumer processing the task.
- **Idle time** — Time since the task was delivered to the worker.
- **Delivery time** — Time when the task was delivered to the worker.
- **Status** — Current task status:
  - **New** — The task has been added to the queue but has not yet been assigned to a worker.
  - **In progress** — The task has been assigned to a worker and is being processed.

Configuration

The widget requires no additional configuration and works immediately after you add it to the dashboard.

Technology radar

The widget visualizes technologies, tools, and practices used in the company, grouped by maturity level: Adopt, Trial, Assess, and Hold.

The widget displays a circular chart with four quadrants, four rings, and a set of items. Each item has a number, name, quadrant, and ring. Configure the radar contents in the widget settings.

Configuration

Name	Required	Description
Quadrants	Yes	Quadrant names
Items	No	List and configuration of up to 200 radar items

Item configuration

Name	Required	Description
Name	Yes	Item name
Number	Yes	Integer from 0 to 9999
Description	No	Item description, displayed as Markdown in the details dialog
Quadrant	Yes	Quadrant containing the item
Ring	Yes	Ring containing the item: Adopt, Trial, Assess, or Hold

Numeric value

The widget displays a specified numeric value.

Configuration

Name	Required	Description	Default
Resource	No	Resource from which required values are extracted when processing the template	—
Numeric value	No	Value displayed in the widget. Templating is supported. Without templating: `100`. With templating: `{{ .entity.properties.id }}`	—

Percentage value

The widget displays a specified percentage value.

Configuration

Name	Required	Description	Default
Resource	No	Resource from which required values are extracted when processing the template	—
Percentage value	No	Value displayed in the widget. Templating is supported. Without templating: `100`. With templating: `{{ .entity.properties.id }}`	—

Repository browser

The widget displays the structure and contents of files in GitLab, Bitbucket, and GitHub repositories.

Configuration

Name	Required	Description	Default
Provider	Yes	Service hosting the repository: GitLab, GitHub, or Bitbucket	—
Branch / Tag	No	Branch name, tag, or commit SHA	`main`
Path	No	Directory path in the repository. Leave empty for the root	—
Recursive	No	Retrieve files recursively from subdirectories	`false`

GitLab configuration

Name	Required	Description	Default
Project ID	Yes	GitLab project ID, for example, `12345`	—

Bitbucket configuration

Name	Required	Description	Example	Default
Project key	Yes	Bitbucket project key, for example, `MYPROJ`	`MYPROJ`	—
Repository ID	Yes	Bitbucket repository ID, for example, `my-repo`	`my-repo`	—

GitHub configuration

Name	Required	Description	Default
Repository owner	Yes	Repository owner: an organization or user. For `https://github.com/example/my-repo`, specify `example`	—
Repository	Yes	Repository name without `.git`. For `https://github.com/example/my-repo`, specify `my-repo`	—

Authorization

Configure authorization separately for each provider:

- GitLab.
- Bitbucket.
- GitHub.

Markdown

The widget displays text written in Markdown.

Configuration

Name	Required	Description	Default
Markdown	Yes	Markdown text rendered in the widget	—

Health checks

Overview

For each resource, you can configure any number of rules that determine the status of its entities. The `Condition` field determines whether:

- all rules must pass (`AllOf`);
- at least one rule must pass (`AnyOf`).

> If at least one check returns an error, the entity status is set to `error`, regardless of the other check results and the condition.

Schedule

The check scheduler runs every minute. For each rule, you can specify a five-field cron expression in the **Schedule** field. The rule then runs only at the specified times. If the field is empty, the rule runs every time the scheduler starts (once a minute).

Example: `0 * * * *` runs the check at the beginning of every hour.

If a rule is not scheduled to run at the current time, its latest check result is used to calculate the entity status.

Logs for the latest checks are available under **Health checks** in the resource menu.

Entity statuses

An entity can have one of four statuses:

- `healthy` — rules are configured, and the entity parameters satisfy them;
- `unhealthy` — rules are configured, but the entity parameters do not satisfy them;
- `unknown` — no rules are configured, or the check cannot run;
- `error` — an error occurred while at least one rule was running.

Click an entity status badge to open a table with rule results and additional information. The table shows only rules that have run at least once.

> The `ENTITY_UPDATED` event is generated only when the entity status changes.

Health check types

Property

A `Property` rule checks whether a specific entity parameter matches a template expression.

The rule configuration contains one parameter: an expression written in Go template syntax.

Expression examples:

- `{{ eq .entity.properties.lifecycle "deployed" }}` — the `lifecycle` property must equal `"deployed"`;
- `{{ lt .entity.properties.vulnerabilities 10 }}` — the `vulnerabilities` property must be less than `10`.

Prometheus

A Prometheus rule checks whether a specified metric meets a configured threshold. The configuration contains a PromQL query that must return a Scalar or a single-value Vector.

Templating is supported. For example:

```
avg(ingress_nginx_detail_request_seconds_sum{location="/{{ .entity.slug }}"})
```

In this example, the entity identifier replaces `{{ .entity.slug }}`.

Configuration parameters

Name	Description	Allowed values
Query	PromQL query for a Prometheus metric	
Operator	Comparison operator applied to the query result and threshold	Equal, NotEqual, LessThan, GreaterThan
Threshold	Threshold compared with the query result	

> Each entity check sends a separate request to Prometheus. Account for this when planning system load.

Authorization

Authorization is configured in the Prometheus external service section.

GitLab Pipeline

A `GitlabPipeline` rule checks whether the latest GitLab pipeline for the selected Ref (branch or tag) has the configured status.

All text fields support templating. For example, specify the following expression in the `Ref` field:

```
{{ .entity.properties.mainBranch }}
```

During the check, the value of the checked entity's `mainBranch` parameter replaces the expression.

Configuration parameters

Name	Description	Allowed values
Project ID	GitLab project ID	
Ref	Branch or tag whose latest pipeline is checked	
Status	Pipeline status considered successful	created, waiting_for_resource, preparing, pending, running, success, failed, canceled, skipped, manual, scheduled

> Each entity check sends a separate request to GitLab. Account for this when planning system load.

Authorization

Authorization is configured in the GitLab external service section.

DefectDojo Findings

A `DefectDojoFindings` rule checks the number of active vulnerabilities at each severity level for a specified DefectDojo product.

The check uses the conditions in the `conditions` block. For each severity, you can specify the expected number of vulnerabilities and a comparison operator, for example, Critical < 5.

All text fields support templating. For example, you can insert the product name from the entity parameters:

```
{{ .entity.properties.defectdojo_product_key }}
```

Configuration parameters

Name	Description	Allowed values
Product name	Product identifier in DefectDojo	
Conditions	Conditions for comparing vulnerability counts by severity	

Conditions

Each condition is an object in the following form:

```
conditions:
  - severity: Critical
    operator: "<"
    value: "5"
  - severity: High
    operator: "<="
    value: "10"
```

Field	Description	Allowed values
Severity	Severity level	Total, Critical, High, Medium, Low, Info
Operator	Comparison operator	==, !=, <, <=, >, >=
Value	Target value for the comparison	

Authorization

Authorization is configured in the DefectDojo external service section.

CodeScoring Vulnerabilities

A `CodeScoringVulnerabilities` rule checks the number of vulnerabilities at each severity level for a specified CodeScoring project.

The check uses the conditions in the `conditions` block. For each severity, you can specify the expected number of vulnerabilities and a comparison operator. Both CVSS2 and CVSS3 metrics are supported.

All text fields support templating. For example, you can insert the project ID from the entity parameters:

```
{{ .entity.properties.codescoring_project_id }}
```

Configuration parameters

Name	Description	Allowed values
Project ID	Project identifier in CodeScoring	
Conditions	Conditions for comparing vulnerability counts by severity	

Conditions

Each condition is an object in the following format:

```
conditions:
  - severity: CRITICAL
    operator: "<"
    value: "5"
    cvss: "cvss3"
  - severity: HIGH
    operator: "<="
    value: "10"
    cvss: "cvss2"
```

Field	Description	Allowed values
Severity	Severity level	CVSS3: CRITICAL, HIGH, MEDIUM, LOW, NONE, UNKNOWN. CVSS2: HIGH, MEDIUM, LOW, NONE
Operator	Comparison operator	==, !=, <, <=, >, >=
Value	Target value for the comparison	
CVSS	CVSS metric version	cvss2, cvss3

Authorization

Authorization is configured in the CodeScoring external service section.

SonarQube Metrics

A `SonarqubeMetrics` rule checks SonarQube project metrics against configured conditions.

The check calls the SonarQube REST API endpoint `/api/measures/component` and compares current metric values with the expected values specified under **Conditions**.

All text fields support templating. For example, you can insert the component key from the entity parameters:

```
{{ .entity.properties.sonarqube_project_key }}
```

Configuration parameters

Name	Required	Description	Allowed values
Project key	Yes	Project identifier in SonarQube	
Branch	No	Project branch from which metrics are read	
Conditions	Yes	Conditions for comparing SonarQube metrics	

Conditions

Each condition is an object in the following form:

```
conditions:
  - metric: coverage
    operator: "<"
    value: "5"
  - metric: bugs
    operator: "<="
    value: "10"
```

Field	Description	Allowed values
Metric	SonarQube metric specified by its metric key	See the metric list in the official SonarQube documentation
Operator	Comparison operator	==, !=, <, <=, >, >=
Value	Target value for the comparison	

See the list of available metrics for the current SonarQube version.

Authorization

Authorization is configured in the SonarQube external service section.

SonarQube Quality Gate

A `SonarqubeQualityGate` rule checks the Quality Gate status of a SonarQube project. The check calls the SonarQube REST API endpoint `/api/qualitygates/project_status`.

The rule returns `true` (passes) if the project Quality Gate has the `OK` status. It returns `false` (fails) for all other statuses: `WARN`, `ERROR`, or `NONE`.

All text fields support templating. For example, you can insert the project key from the entity parameters:

```
{{ .entity.properties.sonarqube_project_key }}
```

Configuration parameters

Name	Required	Description	Allowed values
Project key	Yes	Project identifier in SonarQube	
Branch	No	Project branch whose Quality Gate is checked. If omitted, the main branch is checked	

> Each entity check sends a separate request to SonarQube. Account for this when planning system load.

Authorization

Authorization is configured in the SonarQube external service section.

URL

A `URL` rule checks the availability of an HTTP or HTTPS endpoint. It sends a request with the configured parameters and evaluates the result against one or more Go template conditions. Conditions can check the response code, headers, response body, and entity parameters.

All string fields (URL, request body, and headers) support templating with entity data (`.entity`), for example, `{{ .entity.slug }}`.

Configuration parameters

Name	Required	Description	Examples
URL	Yes	Full request URL	`https://example.com`
Method	No	HTTP request method	GET, POST
Query	No	Query value added to the outgoing request	
Save details	No	If enabled, the check table stores and displays the response body and status (code and headers). If disabled, it shows only the condition, result, and error	
Conditions	No	Go template expressions used to evaluate the response	See below
Multiple conditions	No	`AllOf` requires all conditions to pass; `AnyOf` requires at least one	AllOf, AnyOf
Request body	No	Request body in YAML format. After template substitution, it is converted to JSON and sent	

If no conditions are specified, the response code is checked against `200` by default: `{{ eq .status.code 200 }}`.

Conditions

Conditions can use:

- `.status` — response data: `.status.code` (HTTP code), `.status.status` (status string), `.status.headers` (headers), and `.status.contentLength`. `.status.headers` is a map of header names to value arrays, with names in lowercase. Example: access the first header value with `{{ index (index .status.headers "content-type") 0 }}`; iterate over values with `{{ range index .status.headers "set-cookie" }}...{{ end }}`.

- `.response` — response body with converted types (for a JSON response).
- `.entity` — parameters of the checked entity.

Condition examples:

```
{{ eq .status.code 200 }}
{{ eq (index (index .status.headers "content-type") 0) "application/json" }}
{{ gt .status.contentLength 0 }}
```

Request body example

Specify the **Request body** field in YAML format. For example:

```
id: "{{ .entity.id }}"
name: "{{ .entity.name }}"
```

Result evaluation

The check passes when all conditions pass in `AllOf` mode or at least one passes in `AnyOf` mode. If evaluating a condition returns an error—for example, the response is not JSON but the condition uses `.response`—the check returns an error.

External services

External services configure authentication for external infrastructure systems such as GitLab, Kubernetes, and DefectDojo.

Configure external services under “Administration” → “External services”.

Configuration

An external service has the following parameters:

Parameter	Description
“Name”	External service name displayed in the interface
“Identifier”	Unique human-readable identifier (slug)
“Owner”	User responsible for the external service
“Owner team”	Team that owns the external service
“URL”	Base URL for requests to the external service
“Credentials”	Available credentials, such as tokens, that can be included in requests
“Headers”	HTTP headers automatically added to requests
“Disable SSL verification”	Disables verification of the external service’s SSL certificate, for example when using self-signed certificates. Instead, add the required certificates to trusted certificates
“System credential”	Credential used for scheduled jobs such as data source synchronization and status checks

Using external services

External services can be connected to the following DDP (portal) objects:

- actions;
- widgets;
- data sources;
- entity status checks.

Connect a service on the “Authorization” tab in the settings of the corresponding object.

Each object can explicitly override parameters configured for the external service. For example, if an external service contains an authentication token but an action requires different credentials, specify alternative values. Only the specified parameter is overridden; all other values come from the external service configuration.

Object-specific behavior

Actions

- Multiple external services can be connected to one action.
- One service can be selected as the default.
- The “System credential” parameter is not used for actions.

- When running an action, you can select the external service to use. If you do not select a service, the default service is used. If no default is configured, the first external service in the list is used.

Widgets

- Multiple external services can be connected to one widget.
- One service can be selected as the default.
- A future release will allow users to change the service while viewing a widget.
- The “System credential” parameter is not used for widgets.

Data sources

- Only one external service can be connected.
- If the service has a system credential, it is used by default during synchronization.

Status checks

- One external service can be assigned to a status check.
- If the service has a system credential, it is used for automatic checks.

Authorization headers for external services

When configuring an external service, specify the HTTP headers required for authentication. The following sections list supported external services, authentication methods, and required headers.

> The examples below show possible authentication methods for each service. Some services may support other methods. The portal supports any method that can be passed through HTTP headers.

CodeScoring

Authentication type: API token.

Headers:

Header	Value format
`Authorization`	`<token>`

Example:

```
Authorization: <your-api-token>
```

ClickHouse

Authentication type: Basic Authentication or the `X-ClickHouse-User` and `X-ClickHouse-Key` headers.

Headers:

Header	Value format
`Authorization`	`Basic <base64-encoded-credentials>`
`X-ClickHouse-User`	`<username>`
`X-ClickHouse-Key`	`<password>`

Basic Authentication example:

1. Create the `username:password` string.
2. Encode it in Base64: `echo -n "username:password" | base64`.
3. Add the header:

```
Authorization: Basic <base64-encoded-credentials>
```

Example using ClickHouse headers:

```
X-ClickHouse-User: <username>
X-ClickHouse-Key: <password>
```

DefectDojo

Authentication type: API v2 key token.

Headers:

Header	Value format
`Authorization`	`Token <token>`

Example:

```
Authorization: Token <your-defectdojo-api-v2-key>
```

Bitbucket

Authentication type: bearer token (personal access token).

Headers:

Header	Value format
`Authorization`	`Bearer <token>`

Example:

```
Authorization: Bearer <your-bitbucket-personal-access-token>
```

Docker Registry

Authentication type: Basic Authentication.

Headers:

Header	Value format
`Authorization`	`Basic <base64-encoded-credentials>`

Example:

1. Create the `username:password` string.
2. Encode it in Base64: `echo -n "username:password" | base64`.
3. Add the header:

```
Authorization: Basic <base64-encoded-credentials>
```

GitLab

Authentication type: personal access token or project access token.

Headers:

Header	Value format
`Private-Token`	`<token>`

Example:

```
Private-Token: <your-gitlab-token>
```

For instructions on creating a GitLab token, refer to the GitLab authentication documentation.

GitHub

Authentication type: personal access token.

Headers:

Header	Value format
`Authorization`	`Bearer <token>`

Example:

```
Authorization: Bearer <your-github-token>
```

Create the token in GitHub under “Settings” → “Developer settings” → “Personal access tokens”.

Harbor

Authentication type: Basic Authentication.

Headers:

Header	Value format
`Authorization`	`Basic <base64-encoded-credentials>`

Example:

1. Create the `username:password` string.
2. Encode it in Base64: `echo -n "username:password" | base64`.
3. Add the header:

```
Authorization: Basic <base64-encoded-credentials>
```

Jenkins

Authentication type: Basic Authentication (username and password).

Headers:

Header	Value format
`Authorization`	`Basic <base64-encoded-credentials>`

Example:

1. Create the `username:password` string, where:
   - `username` is the Jenkins username.
   - `password` is the user’s password.
2. Encode the string in Base64: `echo -n "username:password" | base64`.
3. Add the header:

```
Authorization: Basic <base64-encoded-credentials>
```

Jira

Authentication type: Basic Authentication.

Headers:

Header	Value format
`Authorization`	`Basic <base64-encoded-credentials>`

Example:

1. Create the `username:password` string.
2. Encode it in Base64: `echo -n "username:password" | base64`.
3. Add the header:

```
Authorization: Basic <base64-encoded-credentials>
```

Kaiten

Authentication type: API token.

Headers:

Header	Value format
`Authorization`	`Bearer <token>`

Example:

```
Authorization: Bearer <your-kaiten-api-token>
```

Kubernetes

Authentication type: bearer token (service account token or user token).

Headers:

Header	Value format
`Authorization`	`Bearer <token>`

Example:

```
Authorization: Bearer <your-kubernetes-token>
```

Nexus

Authentication type: Basic Authentication.

Headers:

Header	Value format
`Authorization`	`Basic <base64-encoded-credentials>`

Example:

1. Create the `username:password` string.
2. Encode it in Base64: `echo -n "username:password" | base64`.
3. Add the header:

```
Authorization: Basic <base64-encoded-credentials>
```

OpenSearch

Authentication type: Basic Authentication.

Headers:

Header	Value format
`Authorization`	`Basic <base64-encoded-credentials>`

Example:

1. Create the `username:password` string.
2. Encode it in Base64: `echo -n "username:password" | base64`.
3. Add the header:

```
Authorization: Basic <base64-encoded-credentials>
```

Prometheus

Authentication type: bearer token or Basic Authentication.

Headers:

Header	Value format
`Authorization`	`Bearer <token>` or `Basic <base64-encoded-credentials>`

Bearer token example:

```
Authorization: Bearer <your-token>
```

Basic Authentication example:

```
Authorization: Basic <base64-encoded-credentials>
```

SonarQube

Authentication type: bearer token.

Headers:

Header	Value format
`Authorization`	`Bearer <token>`

Example:

```
Authorization: Bearer <your-token>
```

Svacer

Authentication type: bearer token.

Headers:

Header	Value format
`Authorization`	`Bearer <token>`

Example:

```
Authorization: Bearer <your-token>
```

Vault

Authentication type: token authentication.

Headers:

Header	Value format
`X-Vault-Token`	`<token>`

Example:

```
X-Vault-Token: <your-vault-token>
```

Trusted certificates

Trusted certificates allow you to upload root and intermediate certificate authority (CA) certificates or server certificates to Deckhouse Development Portal (DDP, portal). The portal uses them for TLS/SSL verification when connecting to external services over HTTPS, for example when accessing external service APIs from data sources and widgets.

This mechanism provides secure connections to services that use self-signed or corporate certificates without disabling SSL verification.

> Add certificates for Dex, PostgreSQL, and Redis through the module configuration. DDP connects to these internal services before enabling the trusted certificate mechanism.

Configure trusted certificates under “Administration” → “Trusted certificates”.

Configuration

Specify the following parameters when adding a trusted certificate:

Parameter	Description
“Name”	Certificate name displayed in the interface, for example “Corporate CA”
“Certificate (PEM)”	Certificate content in PEM format. Required when creating the certificate

Specify the certificate in PEM format:

```
-----BEGIN CERTIFICATE-----
MIIDXTCCAkWgAwIBAgIJAKL...
...
-----END CERTIFICATE-----
```

After you save the certificate, its card displays data extracted from the PEM content, including the subject, issuer, validity period, and alternative names.

When editing a certificate, the “Certificate (PEM)” field is hidden. For security, the API does not return the saved value. To update a certificate, create a new record and delete the old one if necessary.

Usage

Uploaded trusted certificates are automatically added to the root certificate pool used by DDP Backend for outgoing HTTPS requests. This pool is used for:

- requests to external services from actions, widgets, data sources, and status checks;

- requests to infrastructure system APIs such as GitLab, Kubernetes, Vault, and Prometheus.

If an external service uses a certificate issued by your own or a corporate CA, add the corresponding root or intermediate certificate under “Trusted certificates”. The portal then trusts the connection without disabling SSL verification.

> Instead of disabling SSL verification for an external service with the “Disable SSL verification” option, add the required certificates as trusted. This preserves authentication and connection security.

Changes to the trusted certificate list, including additions, updates, and deletions, apply to new outgoing connections without restarting DDP.

Access permissions

Trusted certificate management is controlled by the following global permissions:

- `read:trusted-certificates` — view the certificate list and details;
- `create:trusted-certificates` — add certificates;
- `update:trusted-certificates` — edit the name of an existing certificate;
- `delete:trusted-certificates` — delete certificates.

For details about the role model, refer to the RBAC documentation.

User guide

Overview

This document is the User's Guide for the Deckhouse Development Portal and is part of the operational documentation for the Deckhouse Development Portal.

Interface

The Deckhouse Development Portal interface consists of the following sections:

- "Home": the portal's landing page. You can place one or more dashboards with widgets here.
- "Catalog": a service catalog for viewing resources, entities, and relationships, and for running actions and scenarios.
- "Self-Service": a section for configuring data sources, actions, webhooks, automations, scenarios, dashboards, and widgets. Access to this section can be restricted by the RBAC model.
- "AI": a section for configuring MCP servers, custom tools, and tool collections. Access to this section can be restricted by the RBAC model.
- "Administration": a section for managing teams, users, access policies, and credentials. Access to this section can be restricted by the RBAC model.

Global search

At the top of the "Catalog" sidebar, a "Search" field lets you quickly find entities across the portal.

Limitations

- Maximum query length: 255 characters.
- Search works only for entities. Resources, teams, and other portal objects are not included in search results.

AI assistant

> ⚠ Experimental feature

The AI assistant is an intelligent helper built into Deckhouse Development Portal (DDP, portal). It answers questions about the portal, analyzes catalog data, and performs tasks using Model Context Protocol (MCP) tools.

The AI assistant uses configurable AI providers to process requests. It supports various language models, including OpenAI GPT, Ollama, and any models available through a compatible REST API.

Connecting an AI provider

> Users configure AI providers in their profiles.

To connect a new AI provider:

1. Open **Profile → AI providers**.
2. Select **Add**.
3. Fill in **Name**, **Model**, **URL**, **Method**, and **Headers**. If necessary, specify the Response field and Request body template.
4. When using tokens in headers, store the credentials and insert them through templating.
5. Select **Save**.

For common API configurations, refer to Configuration examples.

Provider credentials

The credential system securely stores tokens and keys: it encrypts them in the database and inserts them into request headers through templating.

Adding credentials

To add credentials:

1. In the provider creation or editing form, select **Manage credentials**.
2. In the dialog, select **Add credentials**.
3. Enter a **Key** and **Value** pair, for example, `api_key` and its secret.
4. Select **Save**.

Editing credentials

Consider the following:

- You cannot change the key of existing credentials. Delete the entry and create a new one.
- To update a value, enter a new one.
- Saved values are not displayed in the interface.

Templating in headers

Use a substitution instead of a plaintext token:

```
Authorization: Bearer {{ .credentials.api_key }}
```

Here, `Authorization` is the header, `credentials` is the encrypted credential store, and `api_key` is the key defined in the credentials.

Response field

The **Response field** defines the path to the model response text in the JSON API response body, for example, `choices.0.message.content`. If the field is empty, the portal attempts to locate the response text automatically.

> To determine the path, send a test API request, open the response, and locate the field containing the model response text in the JSON body.

Request body template

The **Request body template** field contains the JSON structure sent to the API.

The following variables are available:

- `{{.prompt}}` — User request text.
- `{{.model}}` — Model configured for the provider.

Structure examples

OpenAI-compatible chat:

```
{
  "model": "{{.model}}",
  "messages": [
    {
      "role": "user",
      "content": "{{.prompt}}"
    }
  ],
  "temperature": 0.7
}
```

Alternative message format:

```
{
  "messages": [
    {
      "content": "{{.prompt}}",
      "role": "user"
    }
  ],
  "model": "{{.model}}",
  "stream": false
}
```

Minimal format:

```
{
  "query": "{{.prompt}}",
  "model_name": "{{.model}}"
}
```

> The template must be valid JSON. You can add fields such as `temperature` and `max_tokens`.

Configuration examples

OpenAI API (Chat Completions)

1. **Name** — Any name, for example, `ChatGPT`.
2. **Model** — For example, `gpt-4` or `gpt-3.5-turbo`.
3. **URL** — `https://api.openai.com/v1/chat/completions`.
4. **Method** — `POST`.
5. **Headers** — For example, `Authorization: Bearer {{ .credentials.openai_api_key }}`. Add the `openai_api_key` key under **Manage credentials**.
6. **Response field** — `choices.0.message.content`.
7. **Body template** — Use the first example under **Request body template**.

Ollama (`/api/generate`)

1. **URL** — `http://localhost:11434/api/generate`, or your Ollama address.
2. **Method** — `POST`.
3. **Response field** — `response`.
4. **Body template**:

   ```
   {
     "model": "{{.model}}",
     "prompt": "{{.prompt}}",
     "stream": false
   }
   ```

Custom REST API

- **URL** — Your service endpoint, for example, `https://api.example.com/v1/chat`.
- **Method** — Usually `POST`.
- **Headers** — For example, `Authorization: Bearer {{ .credentials.api_key }}` and `Content-Type: application/json`.
- **Response field** — Path to the text field in your JSON response.

- **Body template** — Base it on the first example under Request body template. Add fields such as `max_tokens` if necessary.

Using the AI assistant

The assistant panel opens on the right. Use the button in the lower-right corner of the screen to open it.

Selecting a provider

The provider list is at the top of the chat panel. If only one provider is available, it is selected automatically.

Chats

Conversations are organized into chats. The chat list and **New chat** are on the left.

Chats have the following constraints:

1. Each user can have up to 20 chats. When you reach the limit, delete old chats from the ⋯ menu.
2. Before the first message, a chat has a default name. After the first question, the question text becomes the chat name. You can select **Rename** to change it.
3. Deleting a chat and its message history is irreversible.

Sending context

Configure **Send context** separately for each chat:

1. **Enabled** — Sends the conversation context for this chat with the request.
2. **Disabled** — Sends only the current message. This uses fewer tokens but does not retain chat context.

With context enabled, long histories are not sent in full with every request. Recent messages are sent in full, while earlier messages are summarized in a separate request to the same provider.

> ⚠ Sending context increases token usage when interacting with the model.

MCP tools

The AI assistant uses built-in portal tools and tools from MCP collections available to the user.

Expand **Available tools** in the chat panel to view each tool's name, type (`internal`, `external`, or `custom`), arguments, and example. Select an example to insert its text into the input field. The model can call multiple tools in one request.

- Built-in tools (`internal`) and their parameters are described in the MCP server documentation.
- To connect MCP servers, create custom MCP tools, and configure collections, refer to MCP management.

MCP management

Use **AI** → **MCP** to configure tools that the AI assistant and built-in MCP server can call on behalf of a user.

Access to **AI** is controlled by the global `view:ai-page` permission. Separate global `read:` and `edit:` permissions control MCP server, tool, and collection configuration. For details, refer to the role model.

Overview

Section	Purpose
MCP servers	Connect upstream MCP servers and synchronize their tool catalogs
MCP collections	Group tools and control user access
MCP tools	Create custom MCP tools without an upstream MCP server
Catalog	View all available tools

The AI assistant displays the following tool types:

- `internal` — Built-in portal tools described in the MCP server documentation.
- `external` — Tools retrieved from a connected upstream MCP server.
- `custom` — Custom MCP tools.

Tools of the `external` and `custom` types can be called only if they belong to an enabled MCP collection available to the user.

MCP servers

Connect an upstream MCP server to import its tools into the portal catalog.

1. Go to **AI** → **MCP** → **MCP servers**.
2. Select **Connect**.
3. On the **General information** tab, specify **Name**, **Identifier**, **Description**, **Owner**, and **Team**.
4. On the **Configuration** tab:
    1. Enable the **Enabled** toggle.
    2. Select a **Transport**: `HTTP` or `SSE`.
    3. Specify the upstream MCP server **URL**.
    4. If necessary, add **HTTP headers** and **Credentials** for authentication.
5. Select **Save**.

After saving, open the server card and select **Synchronize** to load the tool catalog. On the **Tools** tab, enable the required tools and add **Tags** if necessary.

MCP tools

Create a custom MCP tool that the AI assistant calls directly without an upstream MCP server.

1. Go to **AI** → **MCP** → **MCP tools**.
2. Select **Add**.
3. On the **General information** tab, specify **Name**, **Description**, **Owner**, **Team**, and **Tags**.
4. On the **Configuration** tab:
    1. Enable the **Enabled** toggle.
    2. Define the **Argument schema** as a `JSON Schema` with the root type `object`.
    3. If necessary, specify **Path parameter mapping** as a `JSON` object that maps argument names to `{placeholder}` segments in the executor URL.
    4. If necessary, specify **Query parameter mapping** as a `JSON` object that maps argument names to URL query parameter names.
5. On the **Authorization** tab, specify the executor endpoint **URL**, **HTTP method**, **HTTP headers**, and **Credentials**. The URL can contain `{placeholder}` segments for path parameters and credential placeholders such as `{{ .credentials.tenant_id }}`.
6. Select **Save**.

For example, assume the argument schema defines the `space_id` argument, the executor URL is `https://api.example.com/v1/spaces/{spaceId}/boards`, and the path parameter mapping is `{"space_id": "spaceId"}`. If the tool is called with `space_id=42`, the request is sent to `https://api.example.com/v1/spaces/42/boards`.

For `GET` and `DELETE`, arguments not mapped to path parameters are passed only as query parameters. For `POST`, `PUT`, and `PATCH`, arguments not mapped to path or query parameters are passed in the JSON request body.

MCP collections

An MCP collection groups catalog tools and determines which tools are available to AI assistant users and external MCP clients.

1. Go to **AI** → **MCP** → **MCP collections**.
2. Select **Create**.
3. On the **General information** tab, specify **Name**, **Identifier**, **Description**, **Owner**, and **Team**.
4. On the **Configuration** tab:
    1. Enable the **Enabled** toggle.
    2. Under **Tools**, select tools from the available catalog. Each item includes the MCP server identifier in parentheses.
5. Select **Save**.

To let a user call collection tools, open the collection card menu and select **Configure access**. Assign users or teams a role with the `use:mcp-collections` permission. For the permission list, refer to the role model.

The collection owner and super administrator automatically receive access to the collection.

Catalog

The **Catalog** section displays all tools available to the current user based on MCP collections and access permissions. This section is read-only. Edit tools on the **MCP servers**, **MCP tools**, and **MCP collections** pages.

Integration with the AI assistant and MCP server

Tools from MCP collections are used as follows:

- In the AI assistant, **Available tools** displays built-in tools and tools from available collections.
- The portal MCP server returns and calls the same collection tools available to the user based on their RBAC permissions.

MCP server

> ⚠ Experimental feature

The MCP server is a Deckhouse Development Portal (DDP, portal) component that implements the Model Context Protocol (MCP). It enables external AI clients, such as LM Studio and Claude Desktop, to interact with the portal. The server uses JSON-RPC 2.0 and provides tools for working with portal resources and proxying requests to external infrastructure services.

MCP is an open protocol for connecting AI models to external systems. For details, refer to the official MCP website.

> ℹ In addition to built-in portal tools, the MCP server can call tools from MCP collections available to the user. Configure MCP servers, custom MCP tools, and collections under MCP management.

Available tools

The following **built-in** portal tools are available. Tools of the `external` and `custom` types become available after you synchronize the catalog and add them to an MCP collection.

get_resources

Gets a list of resources.

Parameters: None.

Returns: A list of resources.

Example:

```
Get a list of resources
```

---

get_external_services

Gets a list of external services, such as GitLab and SonarQube.

Parameters: None.

Returns: A list of external services.

Example:

```
Get a list of external services
```

---

get_resource_entities

Gets all entities of the selected resource.

Parameters:

Name	Type	Required	Description
`resource_uuid`	String	Yes	Resource UUID

Returns: A list of resource entities.

Example:

```
Get all services and show their names and creation dates
```

---

get_entity

Gets one entity by UUID.

Parameters:

Name	Type	Required	Description
`entity_uuid`	String	Yes	Entity UUID

Returns: Data for one entity.

Example:

```
Get the entity with UUID 3fa85f64-5717-4562-b3fc-2c963f66afa6
```

---

get_entity_relations

Gets entity relations.

Parameters:

Name	Type	Required	Description
`resource_uuid`	String	Yes	Resource UUID
`entity_slug`	String	Yes	Entity identifier

Returns: A list of entity relations.

Example:

```
Get the relations of the "api-gateway" entity in the "Services" resource
```

---

get_external_data

Sends an HTTP request to an external service using the user’s credentials.

Parameters:

Name	Type	Required	Description
`external_service_uuid`	String	Yes	External service UUID
`query`	String	Yes	Request description, for example, “get pipelines for project 123”
`api_path`	String	Yes	API path with request parameters, such as pagination
`method`	String	No	HTTP method. Default: `GET`
`body`	String	No	Request body for POST, PUT, or PATCH as a JSON string

Credentials and headers are taken from the external service settings in the portal.

Returns: The result of the HTTP request to the external service.

Example:

```
Get a list of projects from the external GitLab service
```

---

get_actions

Gets a list of actions.

Parameters: None.

Returns: A list of actions.

Example:

```
Get a list of actions
```

---

get_datasources

Gets a list of data sources.

Parameters: None.

Returns: A list of data sources.

Example:

```
Get a list of data sources
```

---

get_processes

Gets a list of processes.

Parameters: None.

Returns: A list of processes.

Example:

```
Get a list of processes
```

Connecting to the MCP server

LM Studio

1. Get the connection parameters:
    - Sign in to Deckhouse Development Portal.
    - Get an API token under **Profile**.
    - Note the portal URL, for example, `https://ddp.example.com`.
2. Configure LM Studio:
    - Open LM Studio.
    - Open **Settings**.
    - Find **MCP Servers** or **Model Context Protocol**.
    - Select **Add Server**.
3. Configure the server:
    - **Server Name**: `DDP MCP Server`, or another name.
    - **Server URL**: `https://<DOMAIN>/api/v2/mcp`.
    - **Transport**: `HTTP` or `JSON-RPC`.
    - **Authentication**:
        - **Type**: `Bearer Token` or an equivalent that uses the `Authorization` header.
        - **Header**: `Authorization: Bearer <your_api_token>`.
        - **Token**: Enter the portal API token from **Profile**.
4. Verify the connection:
    - Save the configuration.

    - LM Studio connects to the server after saving.
    - After a successful connection, the following tools are available:
        - `get_resources` — Gets a list of resources.
        - `get_external_services` — Gets a list of external services, such as GitLab and SonarQube.
        - `get_resource_entities` — Gets all entities of the selected resource.
        - `get_entity` — Gets one entity by UUID.
        - `get_entity_relations` — Gets entity relations by resource and identifier.
        - `get_external_data` — Sends an HTTP request to an external service using the user's credentials.
        - `get_actions` — Gets a list of actions.
        - `get_datasources` — Gets a list of data sources.
        - `get_processes` — Gets a list of processes.

After connecting, you can use these tools in conversations with models.

All calls use your access permissions.

Connecting other MCP clients

The Deckhouse Development Portal MCP server is compatible with any client that supports MCP over JSON-RPC 2.0.

To connect a client:

1. **Endpoint URL**: `https://your-platform.com/api/v2/mcp`.
2. **Protocol**: JSON-RPC 2.0.
3. **Authentication**:
    - Header: `Authorization: Bearer YOUR_API_TOKEN`.
    - `YOUR_API_TOKEN` is your portal API token from **Profile**.
4. **Method**: POST.

MCP server request example

HTTP headers:

```
Authorization: Bearer your-api-token-here
Content-Type: application/json
```

Request body:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/call",
  "params": {
    "name": "get_resource_entities",
    "arguments": {
      "resource_uuid": "target-resource-uuid"
    }
  }
}
```

MCP server response example

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "content": [
      {
        "type": "text",
        "text": "[{\"uuid\":\"...\",\"name\":\"Service 1\",\"properties\":{...}},...]"
      }
    ]
  }
}
```

Security

Authentication:

- Every MCP server request must be authenticated with an API token from **Profile**. Pass the token in the `Authorization: Bearer <api_token>` header.
- Access permissions match your portal user permissions.

Access permissions:

- Tools use the same permissions as the user.
- If you cannot access a resource, the tool returns an access error.
- Data is filtered according to your RBAC permissions.

Troubleshooting

Cannot connect to the server

If you cannot connect to the server:

- Verify that the URL is correct and ends with `/api/v2/mcp`.
- Verify that the API token is valid.
- Verify that the portal is accessible from your computer.
- Check the firewall and proxy settings.

Authentication error

If authentication fails:

- Verify the token format.
- Verify that the token has not expired.
- Verify that you use the `Authorization: Bearer <api_token>` header.

Tool returns an access error

If a tool returns an access error:

- Verify that your user has permission to access the requested resource.
- Verify the resource name or identifier.
- Ask the portal administrator to verify your access permissions.

No data returned

If no data is returned:

- Verify the request parameters.
- Verify that the resource exists and contains entities.
- Check the portal logs for detailed error information.

Parameters

For each resource, action, or workflow, the DDP administrator can add an unlimited number of parameters of one of the following types:

- “Array” — a list of values;
- “Boolean” — a boolean value;
- “Date” — a date;
- JSON — text in JSON format;
- “Entities” — an entity, one of whose parameters can be selected as the value;
- “Enum” — an enumeration of values with a key and a displayed value;
- “List” — a list of values with the ability to select one of them;

- Markdown — text in Markdown format;
- “Number” — a number;
- “Object” — an arbitrary object in JSON format;
- “Percentage” — a percentage;
- “String” — a string;
- YAML — text in YAML format;
- “Teams” — teams;
- `URL` — a string in URL format;
- “Users” — users.

Resources

Once a parameter has been added for a resource, it is displayed for all entities of that resource, both in their cards and in the catalog table. A default value can be set for each parameter. The value of each parameter can be changed individually for each entity.

Whether parameters are filled in is checked for each entity, and the check result is shown in the header of the entity’s card.

Synchronizing resource parameters

For various reasons, entities may end up with parameters that the resource no longer has. To remove such parameters, use the “Synchronize parameters” button in the resource menu.

When synchronizing parameters:

- the identifier of each resource parameter is retrieved;
- the list of parameters is retrieved for each entity of the resource;
- entity parameters whose name does not match any of the resource’s parameter identifiers are removed from the entity’s specification.

Actions and workflows

Each action and workflow has a user form consisting of parameters that the user must fill in when launching it.

Restrictions

The identifier of each parameter must:

- contain only the characters `a-z`, `A-Z`, digits, or underscores;
- not start with a digit.

Configuration

- “Editable parameter” — for each parameter, you can allow or disallow the user from editing it. If editing is disallowed, the user will not be able to change the parameter’s value when launching actions or workflows, meaning the default value will always be used.
- “Required parameter” — each parameter can be either required or optional. The value of a required parameter cannot be empty when launching actions or workflows. The value of an optional parameter, however, can remain empty without affecting the action’s operation.
- “Hidden parameter” — a hidden parameter is not displayed in entity tables and cards, or when launching actions or workflows.

Parameter types

Date

A parameter of type “Date” can hold a date value with a user-defined format. In the specification, the value is always stored in ISO 8601 format (`YYYY-MM-DDTHH:mm:ss.sssZ`).

Parameter configuration

- “Format” — configures how the date is displayed. If no format is explicitly set, the default format is used: `YYYY-MM-DDTHH:mm:ss.sssZ` . A description of the format configuration is available in the Day.js documentation. The format setting affects:
  - how the date is displayed in entity tables and cards;
  - the parameter’s value when launching actions or workflows.
- “Current date by default” — substitutes the current date as the default value when editing entities, as well as when launching actions or workflows.
- “Default value” — a predefined default value. Not applied if the “Current date by default” toggle is enabled.

> When launching actions or workflows with the current date used as the default, the parameter must not be hidden in the user form. Otherwise, the current date will not be substituted.

When launching actions or workflows, values cannot be substituted into a “Date”-type parameter using Go template.

List

A parameter of type “List” allows selecting a single value from a predefined list.

Enum

A parameter of type “Enum” allows selecting a single item from a predefined list. Unlike the “List” type, the “Enum” parameter uses a key-value pair, where the key is stored in the specification and the value is displayed to the user. This allows the displayed text to be changed without changing the keys.

Templating

Deckhouse Development Portal (portal) supports templating based on Go template: expressions in curly braces are substituted into configuration fields where this is supported (actions, widgets, data sources, etc.). Below are the built-in functions and context variables (`{{ .property.* }}`, `{{ .entity.* }}`, and others).

In addition to the standard functions, the following are available:

- Built-in portal functions.
- Functions from the sprig library.

> Functions from the sprig library are not supported in data sources.

Built-in functions

extractPart

`extractPart` splits the input string `s` by the specified delimiter `delimiter` and returns the part at the specified index `index`. If the index is out of bounds, an error is returned.

Parameters:

- `s` — the input string to split.
- `delimiter` — the delimiter used to split the input string.
- `index` — the index of the part to return after splitting.

Returns:

- A string representing the part at the specified index after splitting the input string.
- If the index is out of bounds, an error is returned.

Example in a template:

```
{{ extractPart "ddp/demo-service" "/" 1 }} // Output: "demo-service"
```

extractLastPart

`extractLastPart` splits the input string `s` by the specified delimiter `delimiter` and returns the last part. If the string is empty or the delimiter is not found, the original string is returned.

Parameters:

- `s` — the input string to split.
- `delimiter` — the delimiter used to split the input string.

Returns:

- A string representing the last part after splitting the input string.

Example in a template:

```
{{ extractLastPart "ddp/demo-service" "/" }} // Output: "demo-service"
```

toJSON

`toJSON` serializes the given value into a JSON string. Accepts any data type. If serialization is not possible, an error is returned.

Parameters:

- `v` — the value to serialize to JSON.

Returns:

- A JSON string representing the input value.
- An error that occurred during serialization.

Example in a template:

```
{{ toJSON . }} // Output: JSON representation of the object in the current context
```

toYAML

`toYAML` serializes the given value into a YAML string. Accepts any data type. If serialization is not possible, an error is returned.

Parameters:

- `v` — the value to serialize to YAML.

Returns:

- A string with the YAML representation of the input value.
- An error that occurred during serialization.

Example in a template:

```
{{ toYAML . }} // Output: YAML representation of the object in the current context
```

fromYAML

`fromYAML` parses a YAML string into a `map` for use in a template (for example, to merge with `dict` or access by key).

Parameters:

- `s` — a string containing YAML.

Returns:

- A `map` with the data from the string.
- An empty `map` if the string is empty.
- An error if the YAML is invalid.

Example in a template:

```
{{ index (fromYAML "first: a\nsecond: b\n") "first" }} // Output: "a"
```

replaceChar

`replaceChar` replaces all occurrences of a specified character in a string with another specified character.

Parameters:

- `s` — the source string in which to perform the replacement.
- `oldChar` — the character to replace.
- `newChar` — the character to replace it with.

Returns:

- The modified string, where all occurrences of `oldChar` have been replaced with `newChar`.

Example in a template:

```
{{ replaceChar "hello/world" "/" "-" }} // Output: "hello-world"
```

toSlug

`toSlug` converts an arbitrary string into a valid identifier form:

- The result contains only lowercase Latin letters and digits, with hyphens between fragments.
- The result has no leading hyphen.
- The result is no longer than 64 characters.
- Characters outside the allowed set are replaced with a single hyphen.
- Leading and trailing hyphens are removed.

Parameters:

- `s` — the source string.

Returns:

- A string in identifier form, or an empty string.

Example in a template:

```
{{ toSlug "My Service/Name!" }} // Output: "my-service-name"
```

filteredItems

`filteredItems` filters an array of items (maps) by a requested key value. Returns an array of items whose value at the specified key matches the target value.

Parameters:

- `data` — the array of data to filter.
- `key` — the key whose value is checked.
- `value` — the target value to compare against.

Returns:

- An array of items whose value at the specified key matches the target value.
- An error, if a problem occurs during filtering.

Example in a template:

```
{{ filteredItems .items "name" "Alice" }} // Output: [{"name": "Alice", "age": 30}]
```

anyOf

`anyOf` checks whether at least one element of the array satisfies the comparison condition.

Parameters:

- `operator` — the comparison operator (eq, ne, lt, le, gt, ge).
- `value` — the target value to compare against.
- `key` — the key whose value is used for comparison.
- `data` — the array of data (maps).

Returns:

- `true` if at least one value from the data array satisfies the condition.

Example in a template:

```
{{ anyOf "eq" "value" "key" .data }} // Output: true if at least one record satisfies
the condition
```

allOf

`allOf` checks whether all elements of the array satisfy the comparison condition.

Parameters:

- `operator` — the comparison operator (eq, ne, lt, le, gt, ge).
- `value` — the target value to compare against.
- `key` — the key whose value is used for comparison.
- `data` — the array of data (maps).

Returns:

- `true` only if all values satisfy the condition.

Example in a template:

```
{{ allOf "gt" 10 "age" .users }} // Output: true if all users are older than 10
```

getFieldValue

`getFieldValue` retrieves a field's value by key from a structure represented as a map.

Parameters:

- `items` — a data structure in the form of a `map`.
- `key` — the key whose value should be retrieved.

Returns:

- The value obtained by the key.
- An error if the structure is missing or the key is not found.

Example in a template:

```
{{ getFieldValue .item "name" }} // Output: the value of the "name" field from
the .item structure
```

findValueInDictArray

`findValueInDictArray` searches an array of dictionaries for an item whose value at the specified key matches the given value, and returns the value of another key.

Parameters:

- `data` — the array of dictionaries (map[string]interface{}) to search.
- `filterKey` — the key used for filtering.
- `filterValue` — the value that the value at filterKey must match.
- `targetKey` — the key whose value should be retrieved from the found item.

Returns:

- The value corresponding to targetKey in the found dictionary.

- An error if the item is not found or targetKey is missing.

Example in a template:

```
{{ findValueInDictArray .items "environment" "test" "url" }}  // Output: the value of
the "url" key from the first found dictionary where "environment" equals "test".
```

generatePassword

`generatePassword` generates a random password with the specified parameters.

Parameters:

- `length` — password length (default: 16).
- `includeUppercase` — include uppercase letters A-Z (default: true).
- `includeLowercase` — include lowercase letters a-z (default: true).
- `includeNumbers` — include digits 0-9 (default: true).

Returns:

- A string with the generated password.
- An error if a password cannot be generated with the specified parameters.

Examples in a template:

```
{{ generatePassword }}                                    // Output: a random 16-
character password
{{ generatePassword 12 }}                                 // Output: a random 12-
character password
{{ generatePassword 8 true true true }}                   // Output: an 8-character
password
{{ generatePassword 10 false true true }}                 // Output: a 10-character
password using only lowercase letters and digits
```

encodeUnicode

`encodeUnicode` converts a string into a sequence of Unicode escape sequences. Each character in the string is encoded in the format `\uXXXX`, where `XXXX` is the four-digit hexadecimal representation of the character's code point.

Parameters:

- `s` — the string to encode into Unicode escape sequences.

Returns:

- A string where each character is represented as a `\uXXXX` escape sequence.

Example in a template:

```
{{ encodeUnicode "Привет" }} // Output: "\u041f\u0440\u0438\u0432\u0435\u0442"
{{ encodeUnicode "Hello" }}  // Output: "\u0048\u0065\u006c\u006c\u006f"
```

decodeUnicode

`decodeUnicode` decodes a string containing Unicode escape sequences back into a regular string. The function handles escape sequences of the form `\uXXXX`, where `XXXX` is a four-digit hexadecimal number representing a Unicode code point.

Parameters:

- `s` — the string with Unicode escape sequences to decode.

Returns:

- The decoded string, where all Unicode escape sequences have been converted to characters.
- An error if decoding fails (for example, due to an incorrectly formatted escape sequence).

Example in a template:

```
{{ decodeUnicode "\u041f\u0440\u0438\u0432\u0435\u0442" }} // Output: "Привет"
{{ decodeUnicode "\u0048\u0065\u006c\u006c\u006f" }}      // Output: "Hello"
```

jwtSign

`jwtSign` creates a signed JWT from the given claims, signing key, and algorithm.

Parameters:

- `claims` — a `map` or a JSON string. Standard fields: “sub”, “iss”, “aud”, “exp”, “iat”, “nbf”.
- `signingKey` — the signing key: for HS256/HS384/HS512, a secret string; for RS256/ES256 and others, the PEM-encoded private key.
- `algorithm` — the signing algorithm: HS256, HS384, HS512, RS256, RS384, RS512, ES256, ES384, ES512, PS256, etc.
- `headers` — JWT headers (optional): a `map` or a JSON string, e.g. “kid”, “cty”. An empty string or empty `map` means no headers are set.

Returns:

- A string with the signed JWT.
- An error for invalid parameters or key.

Examples in a template:

```
{{ jwtSign (dict "sub" "user-123" "iss" "ddp") .credentials.secret "HS256" "" }}
```

Global variables

Global variables are shared variables that can be reused in templating when actions are launched.

To substitute the value of a global variable, use the following construct in a field that supports templates:

```
{{ .global.<slug>.<key> }}
```

where:

- `global` — indicates that a global variable is being accessed.
- `slug` — the identifier of the global variable set.
- `key` — the key of the variable whose value should be substituted.

> ⚠ Global variables are stored in the DDP database in plain text, and their values can be retrieved by users through the web interface. It is not recommended to store sensitive data in global variables.

Configuring global variables

Global variables are configured in the “Self-service” → “Global variables” section.

The following naming rules apply:

- The name of a global variable set cannot be empty.
- The identifier of a global variable set cannot be empty and must meet the following conditions:
  - Contain only the characters `a-z`, `A-Z`, digits, or underscores.
  - Not start with a digit.
- The key of each variable in the set cannot be empty and must meet the following conditions:
  - Contain only the characters `a-z`, `A-Z`, digits, or underscores.
  - Not start with a digit.
- The value of each variable in the set cannot be empty.

Team variables

Team variables can be used in all actions, workflows, and processes.

Team variables are configured in the “Administration” → “Teams” section, in the team editing menu. Each user can edit the variables of the teams they belong to — this can be done in the user’s profile.

To get the value of a team variable, use the following construct:

```
{{ .team.<variable_name> }}
```

When launching an action, the user must select the team whose variables will be substituted.
When launching a workflow or process, the team is selected once — its variables are used in all actions within it.

Action variables

Action parameters

Action parameters are available through the `{{ .property.* }}` context and contain the values passed when the action is launched.

To get the value of an action parameter, use the following construct:

```
{{ .property.<property_slug> }}
```

where:

- `property` — indicates that an action parameter is being accessed.
- `property_slug` — the identifier of the parameter whose value should be substituted.

Parameter identifiers can be viewed on the “User form” tab in the action configuration window.

Usage examples:

```
{{ .property.environment }}   // Value of the "environment" parameter
{{ .property.count }}         // Value of the "count" parameter
{{ .property.url }}           // Value of the "url" parameter
```

Action response

The action response is available through the `{{ .response.* }}` context and contains the data returned after the action is executed.

To get a value from the action’s response, use the following construct:

```
{{ .response.<field_name> }}
```

where:

- `response` — indicates that the action’s response is being accessed.
- `field_name` — the name of the field in the response whose value should be substituted.

For the response format, see the documentation for the specific action, or check it in the DDP interface:

1. Open the action’s menu (the three-dot button on the action card).
2. Select “Action runs”.
3. Open the configuration of one of the action’s runs.

4. Find the “Response” column in the table.

Usage examples:

```
{{ .response.status }}        // Response status
{{ .response.data.id }}       // ID from the response data
{{ .response.headers.auth }}  // Value of the authorization header
```

⚠ The `{{ .response.* }}` context can only be used in fields that describe entity update rules after an action is run.

Credentials

Credentials are available in all actions, workflows, processes, widgets, data sources, and external services through the `{{ .credentials.* }}` context.

To get the value of credentials, use the following construct:

```
{{ .credentials.<credentials_slug> }}
```

where:

- `credentials` — indicates that credentials are being accessed.
- `credentials_slug` — the identifier of the credentials whose value should be substituted.

The `credentials_slug` identifier matches the one specified on the “Authorization” tab, in the “Credentials” section, in the DDP object configuration dialogs.

Usage examples:

```
{{ .credentials.token }}             // Access token
{{ .credentials.username }}          // Username
{{ .credentials.password }}          // Password
{{ .credentials.accessKeyId }}       // Access Key ID for S3
{{ .credentials.secretAccessKey }}   // Secret Access Key for S3
{{ .credentials.apiKey }}            // API key
{{ .credentials.bearerToken }}       // Bearer token
```

Entity

An entity is available in widgets with `Resource` scope, actions, processes, and workflows through the `{{ .entity.* }}` context.

To get the value of an entity field, use the following construct:

```
{{ .entity.<field_name> }}
```

where:

- `entity` — indicates that an entity is being accessed.
- `field_name` — the name of the entity field whose value should be substituted.

Basic entity fields

```
{{ .entity.uuid }}           // Entity UUID
{{ .entity.slug }}           // Entity identifier
{{ .entity.name }}           // Entity name
{{ .entity.description }}    // Entity description
```

Entity parameters

Entity parameters are available through the `{{ .entity.properties.* }}` context and contain the custom parameters configured for a specific entity.

To get the value of an entity parameter, use the following construct:

```
{{ .entity.properties.<property_slug> }}
```

where:

- `entity` — indicates that an entity is being accessed.
- `properties` — indicates the entity's parameters.
- `property_slug` — the identifier of the entity parameter whose value should be substituted.

Usage examples:

```
{{ .entity.properties.projectId }}     // Project ID from the entity's parameters
{{ .entity.properties.branch }}        // Git branch from the entity's parameters
{{ .entity.properties.environment }}   // Environment from the entity's parameters
{{ .entity.properties.apiUrl }}        // API URL from the entity's parameters
{{ .entity.properties.version }}       // Version from the entity's parameters
```

Process parameters

For each process, you can define shared parameters whose values can be used in all actions that are part of the process.

To get the value of a process parameter, use the following construct:

```
{{ .process.<property_slug> }}
```

where:

- `process` — indicates that the process is being accessed.
- `property_slug` — the identifier of the process parameter whose value should be substituted.

The parameter type and default value are set in the process editing interface. For each specific parameter, the user can override the default value when launching the process, if editing the parameter is allowed.

Usage examples:

```
{{ .process.deploymentUrl }}    // Deployment URL from the process parameters
{{ .process.branch }}           // Git branch from the process parameters
{{ .process.environment }}      // Environment from the process parameters
```

Workflow parameters

For each workflow, you can define shared parameters whose values can be used in all actions that are part of the workflow.

To get the value of a workflow parameter, use the following construct:

```
{{ .workflow.<property_slug> }}
```

where:

- `workflow` — indicates that the workflow is being accessed.
- `property_slug` — the identifier of the workflow parameter whose value should be substituted.

The parameter type and default value are set in the workflow editing interface. For each specific parameter, the user can override the default value when launching the workflow, if editing the parameter is allowed.

Usage examples:

```
{{ .workflow.apiEndpoint }}       // API endpoint from the workflow parameters
{{ .workflow.notificationEmail }} // Notification email from the workflow parameters
{{ .workflow.retryAttempts }}     // Number of retry attempts from the workflow
parameters
```

Process store

The store is only available in processes and is used to pass data between steps. Writing is done via rules in actions and the “Template” element (see Process store); reading is done via placeholders in templates.

To read a value, use:

```
{{ .store.<path> }}
```

The path matches the “Target” field in the write rules. Nested keys and array indices are supported:

```
{{ .store.projectId }}
{{ .store.notification.engagements }}
{{ .store.items[0].status }}
```

Process loop context

While a loop body inside a process is running, the `_loop` object is available in templates:

```
{{ .store._loop.item }}    // current collection item or iteration number
{{ .store._loop.index }}   // 0-based index
{{ .store._loop.total }}   // total number of iterations
{{ .store._loop.first }}   // true on the first iteration
{{ .store._loop.last }}    // true on the last iteration
```

For nested loops, the parent context: `{{ .store._loop.parent }}`.

Product information